**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**
**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**
**імені ІГОРЯ СІКОРСЬКОГО»**
**Факультет інформатики та обчислювальної техніки**
**Кафедра автоматики та управління в технічних системах**

«До захисту допущено»
Завідувач кафедри
_____ О.І. Ролік
«___»_____2019 р.

# Дипломний проект

**на здобуття ступеня бакалавра**

**з напряму підготовки 6. 050201 «Системна інженерія»**

**на тему: «Система стиснення відеоінформації»**

Виконав (-ла):
студент (-ка) IV курсу, групи ІА-52
Громова Тетяна Сергіївна _____

Керівник:
Старший викладач Моргаль О. М.
_____

Рецензент:
_____ _____

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2019 рік

**Національний технічний університет України**
**«Київський політехнічний інститут імені Ігоря Сікорського»**
**Факультет інформатики та обчислювальної техніки**
**Кафедра автоматики та управління в технічних системах**

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки – 6. 050201 «Системна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.І. Ролік
«___»_____2019 р.

# ЗАВДАННЯ
## на дипломний проект студенту
## Громовій Тетяні Сергіївні

1. Тема проекту «Система стиснення відеоінформації», керівник проекту старший викладач Моргаль Олег Михайлович, затверджені наказом по університету від «___»_____ 2019 р. №_____

2. Термін подання студентом проекту 19.06.2019.

3. Вихідні дані до проекту

Інформація про будову файлів GIFформату, їх особливості та методи стиснення. Інформація про алгоритми стиснення зображень, їх роботу та приклади реалізації.

4. Зміст пояснювальної записки

1. Огляд і порівняння існуючих реалізацій (основні поняття, порівняння та аналіз використання популярних графічних форматів, порівняння використання відео та анімації, огляд та порівняння базових алгоритмів стиснення).
2. Постановка та алгоритм розв’язку задачі (визначення мети та цілі роботи, розгляд алгоритму LZW, алгоритму JPEG, GIF формату).
3. Розроблений комплексний алгоритм (структура та внутрішня будова файлу GIF формату, структура та будова JPEG файлу, особливості реалізації алгоритму LZW, особливості реалізації алгоритму JPEG).

4. Реалізація (етапи виконання роботи та пропозиція, програмний додаток для перевірки роботи алгоритму, приклади можливої реалізації LZW та JPEG, результати роботи програми, варіанти подальшого розвитку роботи).

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо)

Структура та внутрішня будова файлу GIF формату — кресленик. Структура JFIF файлу — кресленик. Блок-схема алгоритму стиснення LZW — кресленик. Структура JPEG-перетворень — кресленик. Презентація проекту.

7. Дата видачі завдання 10.04.2019.

Календарний план

| № з/п | Назва етапів виконання дипломного проекту | Термін виконання етапів проекту | Примітка |
|---|---|---|---|
| 1 | Отримання завдання | 10.04.2019 | |
| 2 | Збір інформації | 23.04.2019 | |
| 3 | Огляд та порівняння форматів зображень | 30.05.2019 | |
| 4 | Вивчення проблематики | 01.05.2019 | |
| 5 | Дослідження алгоритмів стиснення | 13.05.2019 | |
| 6 | Застосування реалізованих алгоритмів | 20.05.2019 | |
| 7 | Перевірка результатів роботи системи | 25.05.2019 | |
| 8 | Оформлення дипломної роботи | 10.06.2019 | |
| 9 | Отримання допуску до захисту | 04.06.2019 | |

Студент Т. С. Громова

Керівник проекту О. М. Моргаль

# АНОТАЦІЯ

Тема роботи: Система стиснення відеоінформації.

Виконавець: Громова Тетяна Сергіївна

Пояснювальна записка містить: 86 сторінок, 19 рисунків, з яких 4 діаграми, 12 таблиць, 2 додатки, 19 посилань.

Ключові слова: стиснення, формат зображення, графічний формат, алгоритм стиснення, формат GIF, алгоритм JPEG, алгоритм LZW.

Об’єктом розробки є система з комплексного стиснення зображення алгоритмами JPEG та LZW для формату GIF.

Мета роботи: формування пропозиції нового способу покращення GIF формату та GIF анімації, а саме — нового прийому оптимізації анімації шляхом зменшення розміру вхідних кадрів для подальшого впровадження у системи відображення.

У дипломному проекті було досліджено основні поняття заданої тематики — відеоінформації, графічної інформації, форматів зображень, алгоритмів стиснень. Було визначено сфери застосування та огляд наявних рішень. Досліджено роботу популярних алгоритмів стиснення, їх переваги та недоліки. Детальніше розглянуто формат GIF та GIF анімацію.

В результаті проведених робіт з дослідження проблематики сфери, було визначено необхідність оптимізації розміру GIF анімації.

Запропоновано новий комплексний алгоритм, який працює у два кроки, послідовно застосовуючи алгоритм JPEG та LZW. На основі розробленого алгоритму створено програмний додаток, який дозволяє зменшити розмір GIF файлу без значних втрат якості зображення.

Отримані результати можуть бути корисними для використання у всіх сферах застосування формату GIF та GIF анімації, а саме — рекламі та бізнесі, дозвіллі, мистецтві, спілкуванні між людьми та у соціальних мережах.

# ABSTRACT

R&D: Video Compression System

Author: Hromova Tetiana

Work contains: 86 pages, 19 images, 12 tables, 2 attachments, 19 references.

Key words: compression, image format, graphic format, compression algorithm, format GIF, algorithm JPEG, algorithm LZW.

The object of the development is a two-stage image compression system with JPEG and LZW algorithms for the GIF format.

Purpose: to create a proposal for a new way of improving GIF format and GIF animation, that is the new way of optimization of animation by reducing the size of input frames for further implementation in the display system.

In the diploma project the basic concepts of the given subject were studied — video information, graphic information, image formats, compression algorithms. The areas of application and review of available solutions were identified. The work of popular compression algorithms, their advantages and disadvantages is researched. The GIF format and GIF animation is discussed in more detail.

As a result of the work on the study of field issues, the need to optimize the size of GIF animation was identified.

A new complex algorithm is proposed, which operates in two steps, consistently applying the JPEG and LZW algorithms. Based on the developed algorithm, a software application has been created that allows you to reduce the size of a GIF file without significant loss of image quality.

The results can be useful for use in all areas of GIF and GIF animation, for example, advertising and business, leisure, art, people-to-people communication and social networking.

| Номер рядка | Формат | Позначення | Найменування | К-ть листів | Номер екземп. | Примітки |
|---|---|---|---|---|---|---|
| 1 | | | Документація загальна | | | |
| 2 | | | | | | |
| 3 | | | Знову розроблена | | | |
| 4 | | | | | | |
| 5 | А4 | ІА52.070БАК.005 ПЗ | Пояснювальна записка | 86 | | |
| 6 | А3 | ІА52.070БАК.005 Д1 | Структура та внутрішня будова | 1 | | |
| 7 | | | файлу GIF формату | | | |
| 8 | А3 | ІА52.070БАК.005 Д2 | Структура JFIF файлу | 1 | | |
| 9 | А3 | ІА52.070БАК.005 Д3 | Блок-схема алгоритму | 1 | | |
| 10 | | | стиснення LZW | | | |
| 11 | А3 | ІА52.070БАК.006 Д4 | Структура JPEG-перетворень | 1 | | |
| 12 | | | | | | |
| 13 | | | | | | |
| 14 | | | | | | |
| 15 | | | | | | |
| 16 | | | | | | |
| 17 | | | | | | |
| 18 | | | | | | |
| 19 | | | | | | |
| 20 | | | | | | |
| 21 | | | | | | |
| 22 | | | | | | |
| 23 | | | | | | |
| 24 | | | | | | |
| 25 | | | | | | |
| 26 | | | | | | |
| 27 | | | | | | |
| 28 | | | | | | |

| Зм. | Лист | № докум. | Підп. | Дата | |
|---|---|---|---|---|---|
| Розроб. | | Громова Т. С. | | | Літ. Т / Лист / Листів 1 |
| Перев. | | Моргаль О. М. | | | |
| Н. контр. | | | | | |
| Затв. | | | | | |

# Пояснювальна записка
## до дипломного проекту
## на тему: «Система стиснення відеоінформації»

Київ – 2019 рік

**ЗМІСТ**

## ПЕРЕЛІК СКОРОЧЕНЬ

GIF– Graphics Interchange Format, графічний формат растрових зображень;

JPEG — Joint Photographic Experts Group, графічний формат растрових зображень;

RLE — Run-length encoding, алгоритм стиснення кодування довжин серій;

LZW — Lempel-Ziv-Welch algorithm, алгоритм стиснення словниковим методом;

KWE — Keyword Encoding, група алгоритмів стиснення;

BMP — Bitmap, графічний формат растрових зображень;

PNG — Portable Network Graphics, графічний формат растрових зображень;

SVG — Scalable Vector Graphics, графічний формат векторних зображень;

JFIF — Jpeg File Interchange Format, графічний формат для обміну стисненими JPEG файлами;

ЦОС — Цифрова обробка сигналів;

API — Application Programming Interface, програмний інтерфейс додатку;

ДКП — Дискретно-косинусне перетворення;

МК — Матриця квантування.

# ВСТУП

## Мета, актуальність розробки об’єкту проектування

Інформація є невід’ємною частиною людства і з кожним днем її обсяги збільшуються. Разом з тим виникають питання де та як її зберігати, як збільшити швидкість завантаження у вебі. Інженерами пропонуються нові рішення для збільшення ємності сховищ інформації та збільшення пропускних можливостей мереж. Одним із інструментів, який дозволяє нам позитивно впливати та вирішувати ці проблеми, є стиснення даних. Процедура стиснення являє собою перекодування даних, в результаті чого початкові дані зменшуються у розмірі, тобто скорочується їх обсяг та об’єм. Основним поняттям в теорії стиснення інформації є збитковість. Саме її намагаються зменшити під час компресії.

Для досягнення кращих результатів при перекодуванні, розроблено деякі алгоритми стиснення, які орієнтовані на окремі типи даних. Тому що підходи стиснення будуть відрізнятися для текстових даних, зображень, звуку, відео через особливості цих видів інформації. Це означає, що окремі алгоритми будуть проявляти себе лише на визначених даних і будуть абсолютно неефективні при використанні над іншими.

У даному дипломному проекті буде розглянуто стиснення відеоінформації та алгоритми, що з нею пов’язані. Адже даний напрямок у стисненні інформації активно досліджується завдяки збільшенню потреб користувачів до збереження відеоданих. А все через те, що на сьогодні сфери використання відеоінформації збільшуються і такий формат представлення інформації охопив майже усі частини людського життя.

Для стиснення відео виділяють потокові та статичні алгоритми. Це пов’язано з тим, що відео являється потоком статичних кадрів (зображень). Тому його стиснення можна проводити у трьох вимірах, а саме: двовимірне зображення і третій вимір — час. У даному дипломному проекті буде розглянуто саме статичні методи, так як сфера їх застосування ширша — їх можна

застосовувати як до звичайних зображень, так і до відео. А головний акцент буде зроблений на алгоритм LZW, який застосовується у графічному форматі GIF. Пояснимо чим цікава оптимізація даного формату.

Сьогодні інтернет проник у всі сфери людського життя, тому дуже важливо використовувати стиснення та оптимізацію елементів веб-сторінки. Також відомо, що наявність графічного елементу з рухом, приваблює користувачів та зацікавлює їх залишитися на сторінці. Для цього використовують відео та GIF анімації. Є переваги використання для обох елементів, проте точно відомо, що GIF анімація не впливає на швидкість завантаження сторінки, що є надзвичайно важливим показником. Тож розглядатимемо GIF анімацію як альтернативу до використання коротких відео.

Мета даного дипломного проекту — розробка нового засобу обробки зображення для оптимізації стиснення GIF файлів, а саме GIF анімацій для подальшого їх використання у веб-мережі.

Актуальність даної теми визначає те, що графічна інформація поширена у всіх сферах життя і займає набагато більше місця у пам’яті, ніж текстові файли. Тому активно проводяться дослідження алгоритмів стиснення графіки. Ці алгоритми мають певні відмінності від простих алгоритмів стиснення, а саме: використання особливостей людського зору для створення алгоритмів з втратами, використання особливостей структури зображень (збитковість даних у двох вимірах).

**Обгрунтування основних проектних рішень або напрямків досліджень**

У роботі розглядаються формати зображень JPEG та GIF. Формат JPEG являється одним з найпопулярніших форматів на сьогоднішній день. Найчастіше він використовується для стиснення фотографій та графічних зображень, в яких спостерігається велика кількість кольорів та плавні переходи між ними. Алгоритм JPEG широко застосовується у цифровій фотографії та для збереження і передачі зображень в мережі Інтернет. Він значно скорочує розмір зображення,

проте також впливає на якість зображення, адже являється алгоритмом із втратами.

GIF формат реалізується завдяки алгоритму LZW, який являється алгоритмом без втрат. Даний формат використовується в простій графіці, для зображень з чіткими контурами. Також GIF має особливість, що вирізняє його поміж інших форматів – можливість створювати анімації. Саме ця його особливість допомагає формату залишатися популярним і сьогодні.

У ході дипломного проекту розроблено програмний додаток мовою Java, який не потребує попередніх встановлень та запускається з командного рядку, та допомагає швидко втілювати запропонований спосіб оптимізації зображень.

**Можливі галузі застосування результатів роботи**

У даній роботі запропоновано засіб оптимізації GIF анімації за рахунок зменшення об’єму кадрів, які в ній використовуються. Тому отримані результати можуть використовуватися у всіх сферах, де застосовують GIF формат. А також для зменшення об’єму пам’яті, який займають файли на фізичних та хмарних носіях.

Найчастіше користуються GIF форматом та GIF анімацією у мережі Інтернет, у рекламі. Формат GIF широко представлений у сферах реклами та маркетингу, дозвілля, мистецтва, спілкування між людьми та у соціальних мережах. Детальніше про переваги використання GIF формату буде описано у наступних розділах. Проте популярність даного формату графічних зображень пов’язана саме з властивістю GIF містити у собі набір з декількох зображень-кадрів і об’єднання їх в одну анімацію, з зазначенням часу затримки відображення кожного окремого кадру.

Також результати можуть використовуватися у такій галузі, як цифрова обробка сигналів (ЦОС), а саме у частотно-часовому аналізі. Цей розділ ЦОС вивчає компресію зображень, використання і вплив різних алгоритмів стиснення на якість та розмір цифрових зображень. Зменшення об’єму пам’яті, який займає

зображення, позитивно впливає на час передачі зображення мережею, тобто зменшує його. Це проблема, яку намагаються вирішити сучасні інтернет-провайдери для покращення якості своїх послуг.

Для подальшого розвитку комплексного алгоритму можна продовжити роботу над покращенням результатів на кожну з етапів обробки. Завдяки створенню нових прикладів реалізацій алгоритмів LZW та JPEG можливе збільшення їх ефективності — збільшення коефіцієнту стиснення, скорочення часу обробки файлу, збереження якості початкового зображення. В такому випадку застосування покращених реалізацій складових частин комплексного алгоритму у його роботі, дозволить покращити характеристики алгоритму.

Отже, результати даного проекту містять у собі практичну цінність для подальшого використання у багатьох сферах людського життя і можуть бути включені до існуючих систем стиснення зображень. Також зібрана інформація може бути корисною для звичайних користувачів під час вибору формату зображення або застосування алгоритму стиснення. Напрацювання, отримані у дипломному проекті, можуть бути розвинені для створення покращеного комплексного алгоритму.

# 1 ОГЛЯД І ПОРІВНЯННЯ ІСНУЮЧИХ РЕАЛІЗАЦІЙ

## 1.1 Основні поняття

### 1.1.1 Відеоінформація та її сфери застосування

Відеоінформацією вважають потік кадрів (зображень), які змінюються протягом заданого часу з певною частотою. Також можна визначити це поняття як тривимірний масив кольорових пікселів. Координатами являються розширення кадру по горизонталі та вертикалі, а також час, з яким пов’язане відображення кожного окремого кадру. Кадром будемо вважати набір значень пікселів, які розрізняються камерою в окремий проміжок часу. Швидкість відтворення відеосигналу зазвичай складає 25-30 кадрів в секунду.

З появою першої можливості зафіксувати або створити зображення, останні почали активно входити в наше повсякденне життя. Так само і відеоінформація, яка несе в собі також величезну користь, наразі використовується у всіх сферах життя людини. Для прикладу розглянемо такі важливі сфери як:

- Медицина та охорона здоров’я
- Освіта та навчання
- Безпека
- Медіа сфера, дозвілля, мистецтво
- Реклама
- Комунікація та спілкування між людьми, соціальні мережі

У кожній з наведених вище сфер життя, відеоінформація займає свою нішу і виконує певні функції. Так, в медицині відео використовують під час діагностики пацієнта, для дослідження та виявлення причин появи хвороби, для спостереження за перебігом хвороботворних процесів та їхнього впливу на діяльність людини. Наприклад, у апаратах УЗД є можливість спостерігати за діяльністю внутрішніх органів людини через відео зображення на екрані. Також

в палатах деяких пацієнтів встановлюються відеокамери для збору інформації про якість сну та поведінку хворого. У освітній сфері завдяки відео можна показати як відбувається той чи інший процес для кращого його розуміння. Також відео дозволяє проводити навчання в режимі «онлайн», студенти будуть дивитися лекцію як трансляцію, або у записі, знаходячись в тисячах кілометрів від викладача. Безпеку та охорону об’єктів можна організовувати завдяки встановленню камер спостереження для пізнішого перегляду записів або завдяки встановленню веб-камер, за допомогою яких процес стеження буде відбуватися в реальному часі. Сучасну медіа сферу сьогодні майже неможливо уявити за відсутності відеоінформації. Телевізійні програми, YouTube-канали, відео на сторінках інтернет-видань — все це дуже важливі складові нашого інформаційного простору. Під час дозвілля люди переглядають фільми, серіали, різноманітні відео в інтернеті. Також багато хто знімає відео для домашнього використання та задоволення. З розвитком інформаційних технологій, активно почав рости та розвиватися ринок реклами. Виникають нові напрямки, створюються навіть нові науки та професії, і цьому активно сприяє використання відеоінформації. Адже завдяки відео можна не тільки продемонструвати продукт, а також впливати на емоції та думки людини, спонукати її до тих чи інших дій. І на останок, сфера комунікації та спілкування між людьми. Сьогодні буденною справою є спілкування людей через відеозв’язок, запис відеозвернень, використання відео у соціальних мережах, вираження емоцій невеликими анімаційними зображеннями.

У кожній з цих сфер життя є свої специфічні потреби користувачів, а відтак для їх задоволення необхідно використовувати той чи інший формат даних. Далі детальніше розглянемо використання таких популярних форматів, як BMP, SVG, GIF, PNG , JPEG. Також порівняємо переваги та недоліки цих форматів, визначимо їх сфери використання.

### 1.1.2 Типи зображень та види графіки

У комп'ютерній графіці виділяють три головні види графіки — растрову, фрактальна та векторну.

Вони відрізняються способом запису та представлення інформації, мають різні технічні характеристики та сфери застосування. Розглянемо кожен з видів детальніше.

Растрові зображення являють собою двовимірний масив, елементами якого є пікселі. Зображення зберігається у вигляді мозаїки з точок, де кожна точка має свій колір. До растрових зображень відносять цифрові фотографії, відскановані ілюстрації. Відображення на екрані та телевізійні зображення також являються растрами. Але вони мають відмінність від друкованих зображень формою пікселя: для екранів вони квадратні, для друкованих зображень круглі.

У векторній графіці зображення представляється у вигляді сукупності простих геометричних об'єктів. Ці об'єкти називаються примітивами, бо являються дуже простими і базовими для побудови зображення. До примітивів відносять відрізки, дуги, кола. Набір примітивів, що формуються графічний об'єкт називають вектором, звідки і пішла назва виду графіки. Цей вид графіки використовується для чітких зображень, рисунків, графіків та побудови простих елементів для використання у мережі.

Фрактальна графіка заснована на розділі математики, а саме фрактальній геометрії. Фракталом називають фігуру, яка складається з елементів, подібних до цілої фігури. Тобто основною властивістю фракталів є самоподоба. Це дозволяє з частини фігури добудувати повне зображення. Фрактальна графіка застосовується для створення абстрактних зображень, а також для побудови поверхонь (води, піску, хмар).

Нижче виділено головні переваги та недоліки кожного з видів графіки та представлено у вигляді таблиці 1.1:

Таблиця 1.1 — Порівняльна характеристика видів графіки

| Вид графіки | Растрова | Векторна | Фрактальна |
|---|---|---|---|
| Переваги | Дозволяє створити рисунок будь-якої складності. Складні зображення не потребують масштабування та швидко обробляються. Більшість пристроїв вводу-виводу використовують растрову графіку. | Масштабування зображення не викликає змін у якості та появи дефектів. Розмір графічного файлу невеликий. Можна незалежно один від одного редагувати частини зображення. | Вид графіки, який активно розвивається. Допомагає у побудові складних зображень. |
| Недоліки | Великий розмір файлів, навіть для простих зображень. Масштабування погіршує якість зображення. | Підходить не для всіх зображень. Зображення виглядають ненатуральними. | Підходить не для всіх зображень. Значний розмір файлів. Складний процес побудови зображення. |

Можемо зробити висновок, що всі види графіки мають свої особливості, які створюють їх переваги та недоліки. Для оптимального вибору графіки, необхідно враховувати вид зображень, що створюються.

Зображення також поділяють на два типи за наявністю палітри: з палітрою та без. У тих зображеннях, які мають палітру, пікселі виконують важливу функцію, вони зберігають індекс вектору кольорів (палітри). Для зображень без палітри розрізняють зображення в градаціях сірого та у будь-якій системи кольору. У зображеннях з градацією сірого, пікселі зберігають яскравість кожної точки, рівнями сірого.

Також виділяють різні системи представлення кольору, найбільш використовувані серед яких — RGB та YCbCr. У системі RGB кольори передаються різними значеннями інтенсивності червоного, зеленого та синього

компонент. У той же час YCbCr модель складається з трьох компонентів: яскравості Y, компонентів кольоровості — Cb (задає синій відтінок) та Cr (задає червоність). Співвідношення моделей YCbCr та RGB описується рівняннями (1.1) - (1.7) :

$$Y = 0.299R + 0.587G + 0.114B, \quad (1.1)$$

$$Cb = -0.1687R - 0.3313G + 0.5B + 2^{\text{тд}/2}, \quad (1.2)$$

$$Cb = -0.1687R - 0.3313G + 0.5B + 2^{\text{тд}/2}, \quad (1.3)$$

$$Cr = 0.5R - 0.4187G - 0.0813B + 2^{\text{тд}/2}, \quad (1.4)$$

$$R = Y + 1.402Cr, \quad (1.5)$$

$$G = Y - 0.34414\left(Cb - 2^{\text{тд}/2}\right) - 0.71414(Cr - 2^{\text{тд}/2}), \quad (1.6)$$

$$B = Y + 1.722\left(Cb - 2^{\frac{\text{тд}}{2}}\right), \quad (1.7)$$

де тд — точність дискретизації [1].

### 1.1.3 Класи зображень

На сьогоднішній день ще не існує універсального алгоритму стиснення та формату для усіх видів графіки. Різні алгоритми стиснення та формати пристосовані до різних видів зображень. Вони будуть краще розкривати свої переваги та характеристики для властивих їм зображень. Для того, щоб можна було розрізняти зображення, виділяють деякі класи. Тож класом зображення вважатимемо сукупність зображень, стиснення та обробка яких даватиме якісно однакові результати після застосування до них певного алгоритму. Таким чином виділимо чотири основні класи зображень:

- Перший клас: зображення з невеликою кількістю кольорів та великими областями, заповненими одним кольором. До цього класу відносяться зображення, в яких відсутні плавні переходи кольорів. Наприклад, гістограми, графіки, діаграми.

- Другий клас: зображення з плавними переходами кольорів. Сюди відносять зображення, які створені за допомогою комп’ютера.
- Третій клас: фотореалістичні зображення. До цього класу відносять фотографії, відскановані фото.
- Четвертий клас: фотореалістичні зображення з використанням та накладанням ділової графіки. Ці зображення використовують у рекламі. [2]

Для того, щоб порівнювати роботу різних алгоритмів та використання різних форматів, необхідно вказувати для яких класів зображень вони застосовуються. Адже різні алгоритми по-різному показують себе на окремих класах.

### 1.2 Порівняння та аналіз використання основних графічних форматів

На сьогоднішній день створено багато різних графічних форматів. Невпевнений користувач може розгубитися під час вибору потрібного формату для свого зображення. У таблиці 1.2 запропоновано деяку інформацію для полегшення прийнятті рішення щодо вибору формату:

Таблиця 1.2 — Вибір формату зображення

| | Фотографії | Графіка, логотипи, іконки |
|---|---|---|
| Найкращий вибір | WebP, JPEG(з оптимальним ступенем стиснення) | SVG, PNG, WebP, GIF |
| Найгірший вибір | GIF, SVG | JPEG |
| Найкраща якість | JPEG (мінімальне стиснення), WebP,PNG | PNG, SVG, WebP, GIF |
| Найменший розмір файлу | JPEG (максимальне стиснення), WebP | SVG, GIF |

В даній таблиці представлені різні формати, проте серед усіх можемо виділити найбільш популярні, а саме: BMP, SVG, GIF, PNG, JPEG. Усі вони мають свою історію створення та певні особливості, що вирізняють їх поміж усіх

інших. В таблиці 1.3 представлено основні характеристики даних графічних форматів:

Таблиця 1.3 — Властивості форматів зображень

| Формат | Максимальне число бітів/піксель (глибина кольору) | Максимальна кількість кольорів | Максимальний розмір зображення (у пікселях) | Методи стиснення | Вид графіки |
|---|---|---|---|---|---|
| BMP | 24 | 16777216 | 65535x65535 | RLE | растрова |
| SVG | - | 147 | - | LZ77, Huffman | векторна |
| GIF | 8 | 256 | 65535x65535 | LZW | растрова |
| PNG | 24 | 281474976710656 | 65535x65535 | JPEG | растрова |
| JPEG | 24 | 16777216 | 2147483647x2147483647 | LZ77 | растрова |

Як бачимо, всі ці формати відрізняються багатьма параметрами. Для вибору найкращого формату необхідно враховувати сферу та предмет використання зображення.

Тому запропонуємо також алгоритм визначення оптимального зображення, спираючись на властивості форматів.

На рисунку 1.1 зображено дерево для ухвалення рішень щодо вибору формату зображення:

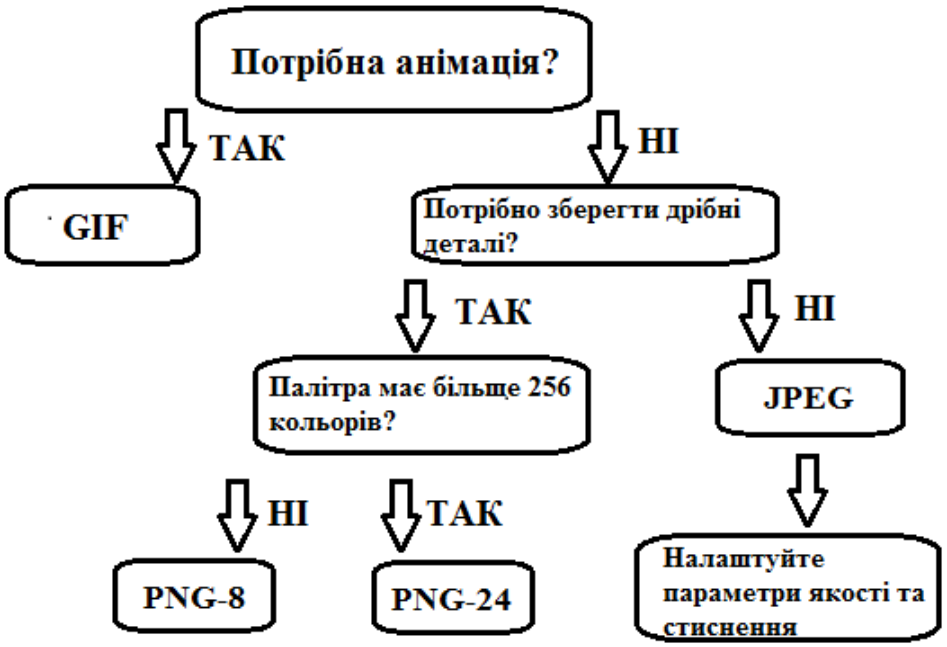


Рисунок 1.1 — Дерево рішень для вибору формату зображення

Відповідаючи на ці запитання приходимо до листків дерева з назвами формату, що буде найкраще підходити до запитів користувача.

Далі розглянемо детальніше особливості популярних форматів та сфери їх застосування.

### 1.2.1 Формат BMP

Першим розглянемо формат BMP. Це формат в якому зображення записується в початковому стані, таким як воно зберігається в пам’яті комп’ютера. У цьому форматі відсутнє стиснення інформації, доступне збереження тільки одного шару зображення. Тому цей формат рідко використовується і призначений для зберігання великих нестиснених фалів у системі Windows. Даний формат був розроблений компанією Microsoft, через це його підтримують велика кількість програм, завдяки спрощеній структурі, а також тому що його підтримка була інтегрована в операційні системи OS/2 та Windows. Через відсутність стиснення, формат майже не використовується в

інтернеті, а це означає, що сфери його використання сьогодні різко скорочуються.

### 1.2.2 Формат SVG

Всі формати, які розглядаються у цьому розділі, стосуються растрових зображень. Формат SVG являється форматом для векторної графіки. Головними елементами такої графіки виступають геометричні об’єкти, які включають у себе криві, лінії, фігури. Особливістю даного формату є те, що графіку, яку розміщено на сайті можна описати текстово розширеною мовою розмітки (XML). Це приваблює використовувати формат у всіх сферах, пов’язаних з вебом. Адже можна швидко змінювати параметри зображення з допомогою мови CSS, для цього можна використати звичайний текстовий редактор. Тому багато експертів вважають, що саме за цим форматом стоїть майбутнє веб-дизайну. Варто зазначити, що використання цього формату у фотографіях неможливе, що різко скорочує його популярність серед звичайних користувачів.

### 1.2.3 Формат GIF

GIF формат раніше займав перше місце по популярності використання. Попередньо розглянутий формат PNG виник йому на зміну, через появу проблем з патентом на використовуваний у форматі GIF алгоритм. GIF використовує алгоритм стиснення без втрат, тому зберігає якість зображення і відсутність появи дефектів. GIF також підтримує прозорість. Проте особливістю формату є те, що він підтримує палітру на 256 кольорів, що значно скорочує його можливості представлення графіки. Тому в більшості випадків його використання змінюється використанням PNG формату. Але незважаючи на свої недоліки GIF все ще використовується, так як має підтримку анімації. Саме ця його особливість призвела до того, що GIF займає свою нішу у таких сферах як реклама, дозвілля, мистецтво та у спілкуванні між людьми в соціальних

мережах. В наступних розділах ми детальніше розглянемо у яких випадках краще звертатися до цього формату зображень.

### 1.2.4 Формат PNG

Формат PNG чудово підходить для фотографій, при цьому зберігаючи високу якісь зображення. Також даний формат підтримує прозорість, тому активно використовується дизайнерами, коли їм потрібно скористатися цією перевагою, не жертвуючи якістю графіки. Формат поділяють на дві версії: PNG-8 та PNG-24. Різниця між версіями полягає в тому, що PNG-8 використовує 256 кольорів, а PNG-24 може оперувати 16 мільйонами. Проте це також означає, що файли PNG-8 будуть мати менший розмір. PNG формат використовується у сфері освіти та дозвілля для створення та збереження рисунків, креслень, символьних картинок (таких як нотні записи, математичні чи будь-які формули, характерні символи та алфавіти). При чому для зображень з невеликою кількістю кольорів та для яких треба отримати менший розмір, використовують версію формату PNG-8.

### 1.2.5 Формат JPEG

JPEG є дуже розповсюдженим форматом, особливо у домашньому використанні. Цьому сприяє те, що формат використовує алгоритм стиснення з втратами, який оснований на особливостях людського зору. Тому розмір файлів значно скорочується, а якість залишається прийнятною для використання у побуті. JPEG активно використовують для цифрових зображень, а також у вебі. Цей формат чудово підходить для графічних даних, якщо нам не потрібна висока якість зображення. Він підтримується усіма браузерами, програмами-переглядачами. Отже, цей формат широко представлений у сфері освіти, медіа, дозвілля, комунікації. У сфері безпеки використовують IP-камери, які підтримують передачу зображень (скріншотів) за протоколом HTTP в форматі JPEG. У сфері медицини дуже важливу роль грає якість зображення, тому тут

JPEG не використовується. Проте для задоволення цих потреб були розроблені, в доповнення основного формату, такі формати як JPEG-LS, Lossless JPEG та JPEG 2000.

### 1.2.5 Використання форматів у мережі Інтернет

Оскільки інтернет сьогодні являється невід'ємною частиною життя, пропонуємо розглянути згадані формати з точки зору популярності їх використання у мережі. Нижче наведено офіційні статистичні дані з сайту W3Techs.com. [3]

Розшифровувати таблицю 1.4 слід так: 100% — уся кількість сайтів мережі інтернет, відсотки у таблиці означають на якій кількості з цих сайтів використовують той чи інший формат.

Таблиця 1.4 — Статистичні дані використання популярних форматів зображень

| | 2018 1 Тра | 2018 1 Чер | 2018 1 Лип | 2018 1 Сер | 2018 1 Вер | 2018 1 Жов | 2018 1 Лис | 2018 1 Гру | 2019 1 Січ | 2019 1 Лют | 2019 1 Бер | 2019 1 Кві | 2019 1 Тра |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PNG | 74.8% | 74.5% | 74.4% | 74.4% | 74.4% | 74.4% | 74.4% | 74.6% | 74.7% | 74.8% | 74.8% | 74.9% | 74.9% |
| JPEG | 73.2% | 73.0% | 72.9% | 72.9% | 72.9% | 72.8% | 72.7% | 72.7% | 72.7% | 72.6% | 72.5% | 72.4% | 72.4% |
| GIF | 31.5% | 30.9% | 30.2% | 29.8% | 29.4% | 28.9% | 28.4% | 28.1% | 27.9% | 27.6% | 27.3% | 27.0% | 26.7% |
| SVG | 11.1% | 11.8% | 12.3% | 12.8% | 13.2% | 13.7% | 14.1% | 14.6% | 15.1% | 15.6% | 16.0% | 16.5% | 16.9% |
| BMP | 0.2% | 0.2% | 0.2% | 0.2% | 0.2% | 0.2% | 0.2% | 0.2% | 0.2% | 0.2% | 0.2% | 0.2% | 0.2% |

На рисунку 1.2 [4] представлено позиції щодо популярності використання для форматів BMP, SVG, GIF, PNG, JPEG у вигляді розміщення на площині, яку складають прямі кількості сайтів та швидкості трафіку цих сайтів.

Дані подані за ситуацією на ринку за травень 2019 року.


Рисунок 1.2— Графічне зображення статистики використання форматів в інтернеті

Даний рисунок підтверджує наведені вище тези щодо популярності та використання різних форматів зображень. Проаналізувавши ці статистичні дані, можемо визначити, що GIF формат поступається таким форматам як JPEG та PNG у поширенні використання, проте він більше використовується в інтернеті на сайтах з вищим трафіком.

### 1.3 Порівняння та аналіз використання відео та анімації

Як було зазначено, у даній роботі увага буде приділятися оптимізації GIF формату. Так як головні переваги GIF анімацій спостерігаються саме у вебі, порівняємо анімації та відео крізь призму використання їх на веб-сайтах.

Відео:

- чудово демонструють продукт та послуги, заряджають користувачів емоціями (для реклами);
- залучають відвідувачів сайту до активної взаємодії;

- вимагають багато часу на створення;
- якісні відео потребують залучення більшої кількості грошових ресурсів: для апаратури, програмного забезпечення;
- збільшують час завантаження сторінки та не підходять для перегляду з поганим мережевих з'єднанням.

GIF анімації:

- економічно ефективні, менше затрат часу та грошей на створення;
- легко редагувати;
- звертають на себе увагу користувачів;
- для створення потрібні лише статичні зображення;
- не впливають на швидкість завантаження сторінки;
- мають колірні обмеження палітри.

Порівнявши ці два види представлення інформації, можна зробити висновок, що GIF анімації мають більше переваг для використання на інтернет ресурсах завдяки своїй економічності, простоті у виконанні, а також через те, що вони не сповільнюють завантаження сторінки.

### 1.4 Огляд базових алгоритмів стиснення

Необхідно чітко розрізняти поняття «формат» та «алгоритм стиснення», адже це різні речі. У форматах файлів використовують алгоритми стиснення для архівації даних, проте формат може реалізовуватися і без компресії. В той самий час, один алгоритм у різних модифікаціях може бути реалізованим у різних форматах. Наприклад, алгоритм RLE використовується у форматах BMP, TIFF, PCX. При чому BMP формат може застосовуватися і без компресії. В такому випадку, можна визначити, що якщо в певному форматі файли мають великий розмір, це не означає, що алгоритм стиснення, який у ньому використовувався, погано стискає. Причиною може бути невдала реалізація алгоритму,

налаштування параметрів алгоритму, кількість кольорів палітри, клас зображення.

### 1.4.1 Класифікація алгоритмів стиснення

Алгоритми стиснення використовують для перекодування даних, для того щоб усунути їх збитковість, зменшити розмір. В залежності від того як відбувається цей процес та як він впливає на вихідні дані, виділяються два типи алгоритмів — без втрат та з втратами.

При чому алгоритми з втратами доцільно використовувати лише для графічної, відео та звукової інформації, адже при їх застосуванні деякі частини файлу спотворюються та спрощуються. Саме за рахунок видалення певних елементів відбувається скорочення об’єму файлу. Проте ми все ще можемо розрізнити подану інформацію. Це пов’язано з тим, що людина завдяки особливостям своїх органів чуттів може коректно сприймати інформацію низької якості.

В той же час, якщо застосувати алгоритм з втратами до текстових даних, можливе спотворення слів чи їх вилучення, і тоді текст втратить свою цінність.

Алгоритми стиснення без втрат не впливають на інформативність файлу, проте дають гірші результати стиснення. Так як головну функцію, яку вони виконують — це перекодування даних для представлення їх у більш компактному вигляді, що не завжди може бути можливим та ефективним.

Варто додати, що алгоритми з втратами також називаються необоротними, а без втрат — оборотними.

На рисунку 1.3 зображено класифікацію алгоритмів стиснення та приналежність окремих алгоритмів до різних типів.


Рисунок 1.3 — Класифікація алгоритмів стиснення

Інколи алгоритми, засновані на статистичному методі називають алгоритмами частотного аналізу, так як вони зважають на частоту появи символів у повідомленні. Поточні алгоритми відносять до алгоритмів кореляційного аналізу, їх також називають словниковими. Також можна виділити основні ідеї алгоритмів такими фразами: на основі статистичного методу — «Скорочуй усе, що зустрічається часто», поточні алгоритми — «Повторюй старе».

### 1.4.2 Вимоги до алгоритмів

Деякі програми, які використовують алгоритми компресії, створені таким чином, щоб користувачі могли обрати оптимальний алгоритм в залежності від своїх потреб. Для цього вони можуть визначати характер використання зображень і пропонувати застосувати той чи інший алгоритм.

Серед вимог, які потрібно враховувати під час вибору алгоритму, виділяють такі:

- ступінь стиснення;
- якість зображення;
- швидкість стиснення;
- швидкість декомпресії;
- масштабування зображень;
- можливість відображення зображення низької роздільної здатності (використовує початок файлу);
- стійкість до помилок;
- врахування класу зображення;
- підтримка редагування;
- вартість апаратної реалізації [2].

Прокоментуємо деякі вимоги. Високий ступінь стиснення та висока якість зображення являються взаємовиключними характеристиками, адже виконання однієї вимоги викликає протиріччя до виконання другої. Висока швидкість стиснення також не може досягатися за виконання однієї з попередніх вимог, так як їх виконання потребує великих затрат часу. Вимога щодо масштабування зображень означає, що зображення повинно легко змінювати свої розміри в залежності від розмірів вікна програми, без погіршення якості та появи небажаних артефактів. Можливість показати зображення низької роздільної здатності необхідна для зображень, які використовуються у вебі — на сторінці, яка все ще завантажується, може відображатися зображення для перед перегляду. Вимога щодо стійкості алгоритму до помилок дуже важлива для передачі файлів.

Врахування класу зображення допомагає обрати оптимальний алгоритм, використовуючи особливості та специфіку зображення. Вимога підтримки редагування означає, що файл не повинен втрачати якість після його повторного збереження (як це відбувається в JPEG).

### 1.4.3 Критерії порівняння алгоритмів

З різноманіття вимог, які були описані вище, випливає те, що використання алгоритмів залежить від умов та від класу зображення. Тому неможливо створити універсальне порівняння усіх відомих алгоритмів, адже їхні характеристики будуть відрізнятися в залежності від ситуації. Проте виділимо декілька основним критеріїв оцінки та порівняння. До них відносяться:

- ступінь стиснення (гірша, середня, найкраща);
- клас зображень;
- симетричність (відношення характеристики алгоритму кодування до характеристики декодування);
- якість (втрати якості);
- особливості алгоритму.

Зазначимо, що при порівнянні алгоритмів також необхідно враховувати їх реалізацію. Адже один алгоритм може мати десятки різних реалізацій, які будуть мати характеристики.

### 1.4.4 Алгоритм Шеннона-Фано

Алгоритм Шеннона-Фано являється одним із перших алгоритмів стиснення. В його основі лежить проста ідея: представлення більш частотних символів за допомогою більш коротких кодів. При цьому коди, які отримуються будуть мати властивість префіксності, тобто жоден з кодів не буде початком ніякого іншого. Властивість префіксності забезпечує, що кодування буде взаємно однозначним.

Представимо короткий ланцюжок дій для виконання алгоритму:

1. Розбиття алфавіту на дві частини, щоб суми ймовірностей їхніх символів були максимально близькі одна до одної.
2. Додавання нулів у префіксні коди першої частини символів та додавання одиниць до другої частини.

3. Виконання кроків 1-3 рекурсивно для кожної частини, в якій не менше двох символів.

Розглянемо приклад кодування рядку «ГГГГТСНРПА». Отримуємо таке дерево Шеннона-Фано (рисунок 1.4):

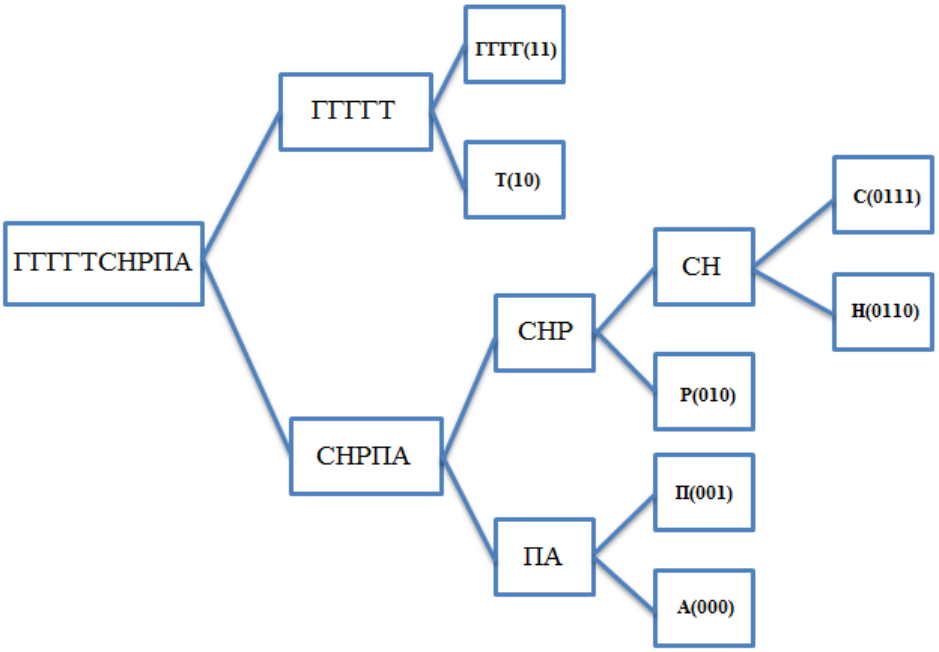


Рисунок 1.4 — Приклад кодування алгоритмом Шеннона-Фано

Без використання кодування повідомлення з прикладу займатиме 40 біт, якщо кожен символ кодується 4 бітами. З використанням алгоритму Шеннона-Фано, закодоване повідомлення займатиме 27 біт.

Незважаючи на те, що алгоритм досить простий, він має певні недоліки. Найбільшим недоліком є те, що кодування є неоптимальним. Так, розбиття на кожному кроці є оптимальним, проте алгоритм не гарантує оптимальність кодування вцілому.

Покращити отримані результати можна за допомогою використання алгоритму Хаффмана, який буде розглянуто нижче.

### 1.4.5 Алгоритм RLE

Алгоритм кодування довжин серій (Run-Length Encoding ) вважається одним з найпростіших та найбільш розповсюджених алгоритмів стиснення даних. Його основна ідея полягає у заміні послідовностей повторюваних символів цим самим символом та кількістю його повторів (коефіцієнтом повторення) [5].

Наприклад нехай дано такий рядок символів «RRRLLLLLLLLLLEEEEEE». З допомогою алгоритму кодування довжин серій він буде перетворений на такий запис «3R10L6E». Очевидно, що такий запис більш компактний і потребуватиме менше пам’яті для зберігання. Проте відразу можна помітити проблему: якщо в початковому рядку буде дана послідовність неповторюваних символів, то алгоритм буде неефективним і навіть збільшувати розмір файлу. Тобто рядок «RLERLERLE» перетвориться у запис «1R1L1E1R1L1E1R1L1E».

Для реалізації RLE алгоритму для графічної інформації, необхідно оброблюване зображення розкласти у ланцюжок з байт за рядками растру. Далі стиснення відбувається за рахунок наявності у вихідному зображенні ланцюжків однакових байт. Ці ланцюжки заміняються парами [лічильник повторів, значення], що зменшує збитковість даних.

Існують деякі модифікації алгоритму, які дозволяють вирішити проблему з ефективністю, якщо зустрічається велика кількість неповторюваних символів. Для цього можна зробити так, що байт, який кодує кількість повторів, буде також зберігати інформацію про наявність повторів. Наприклад, якщо перший біт дорівнює 1, то наступні сім вказуватимуть на кількість повторів відповідного символу, а якщо перший біт дорівнює 0, то наступні біти вказують на кількість символів, які потрібно брати без повторів.

Алгоритм розрахований на зображення з великими областями, заповненими одним кольором. Сюди відносять так звану ділову графіку: гістограми, діаграми, графіки. При застосуванні до цього виду зображень алгоритм буде давати високу ефективність. У випадку коли алгоритм

застосовується до кольорових фотографій, буде спостерігатися збільшення розміру файлу.

Однією з головних переваг алгоритму RLE є швидкість, також він не потребує додаткової пам’яті при роботі, як її потребують словникові методи стиснення. Алгоритм використовується у форматах PCX, TIFF, BMP. Дає такі коефіцієнти стиснення: найкращий 1/32, середній 1/2, найгірший 2/1.

Також цікавою особливістю алгоритму є те, що у форматі PCX для деяких зображень ступінь стиснення може бути підвищений за рахунок зміни порядку кольорів у палітрі.

### 1.4.6 Алгоритм Хаффмана

Алгоритм Хаффмана відомий ще з 60х років, проте не втратив своєї важливості та актуальності і досі. Ідея, яка покладена в його основу досить проста та заснована на частоті появи символу у послідовності. Тому цей алгоритм відносять до алгоритмів частотного аналізу. Варто додати, що символом вважається деякий повторюваний елемент початкового рядка — як друкований знак, так і будь-яка бітова послідовність. Отже, замість того, щоб кодувати усі символи однаковою кількістю бітів, кодування відбувається таким чином: символи, які зустрічаються частіше, кодуються меншим числом біт, а символи, які зустрічаються рідше — навпаки більшим. [6] Ця ідея заснована на спостереженні, що деякі символи зі стандартного набору у довільному тексті можуть зустрічатися частіше середнього періоду повтору, а інші рідше. Відповідно, що якщо використати короткі послідовності бітів для розповсюджених символів, а для запису нечасто використовуваних довгі, то сумарний об’єм файлу зменшиться. Результатом такої систематизації даних буде двійкове дерево.

Алгоритм являється двічі прохідним, тобто реалізується в два етапи. На першому етапі будується частотний словник та генеруються коди. Під час другого проходження безпосередньо відбувається кодування.

Розглянемо приклад. Нехай дано рядок «cool fool feel!», який займає 15*8=120 біти пам’яті. Для того щоб отримати код для кожного символу на основі його частотності, необхідно побудувати бінарне дерево. В дереві кожен листок буде визначений символом. Символи з меншою частотою появи будуть розташовані далі від кореня, ніж символи з більшою, тобто дерево будується від листів до кореня.

Вирахуємо частоти усіх символів (таблиця 1.5):

Таблиця 1.5 — Частота появи символів повідомлення

| Символ | Частота |
|---|---|
| “c” | 1 |
| “o” | 4 |
| “l” | 3 |
| “ ” | 2 |
| “f” | 2 |
| “!” | 1 |
| “e” | 2 |

Далі виділимо вузли бінарного дерева для кожного знаку і додамо їх в чергу за збільшенням частоти (таблиця 1.6):

Таблиця 1.6— Визначення вузлів дерева

| | | | | | | |
|---|---|---|---|---|---|---|
| “!” | “c” | “ ” | “f” | “e” | “l” | “o” |

Далі беремо два перших елементи з черги та зв’язуємо їх, створюючи новий вузол в дереві. Ці елементи будуть потомками вузла, а пріоритет вузла буде дорівнювати їх сумі. Після цього додаємо отриманий новий вузол в чергу.

Повторимо ці операції та отримаємо дерево, причому для отримання коду кожного символу, необхідно пронумерувати гілки. Якщо перехід відбувається вліво, то гілка позначається 0, якщо вправо — 1.

Проходячи ці кроки отримали таке дерево для кодування, яке зображено на рисунку 1.5:

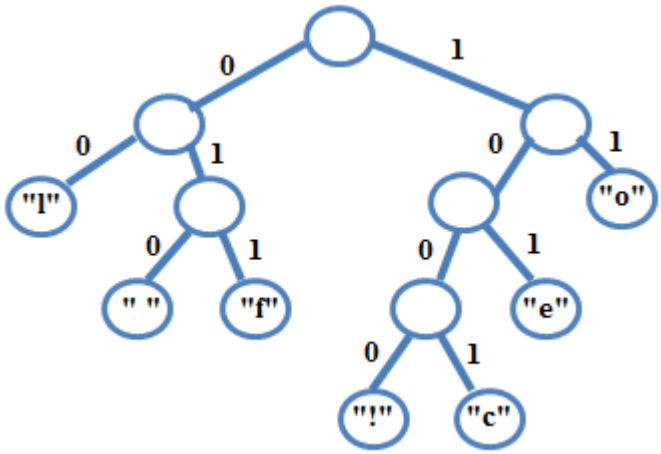


Рисунок 1.5 — Дерево для кодування Хаффмана

На основі дерева створимо таблицю 1.7 для кодування символів:

Таблиця 1.7 — Таблиця для кодування символів

| Символ | Код |
|---|---|
| “c” | 1001 |
| “o” | 11 |
| “l” | 00 |
| “ ” | 010 |
| “f” | 011 |
| “!” | 1000 |
| “e” | 101 |

Таким чином вхідна послідовність у вигляді рядку «cool fool feel!» після кодування виглядатиме таким чином: 1001 11 11 00 010 011 11 11 00 010 011 101 101 00 1000. Послідовність у такому вигляді займає лише 40 біт, що в три рази менше початкового розміру.

Отже підсумуємо, алгоритм Хаффмана володіє властивістю префіксності та мінімальною збитковістю. Для отримання кодів необхідно виконати такі дії:

1. Представити усі символи алфавіту у вигляді вільних вузлів бінарного дерева, при чому вага вузлів пропорційна частоті символу в повідомленні.
2. Вибрати два вузли з мінімальною вагою та створити новий батьківський вузол з вагою, що дорівнює сумі обраних вузлів.

3. Обрані вузли видалити зі списку вільних вузлів, а створений батьківський вузол додати у список.
4. Повторити кроки 2 та 3 до тих пір, доки у списку залишатиметься більше одного вільного вузла.
5. Кожному символу алфавіту присвоїти префіксний код, створений на основі побудованого дерева.
6. Закодувати повідомлення отриманими кодами.

Для стиснення зображень використовується модифікація алгоритму — алгоритм Хаффмана з фіксованою таблицею. Цей алгоритм використовується при стисненні чорно-білих зображень. Заснований на поняттях серії (набору точок зображення одного кольору, що йдуть одна за одною) та довжини серії. В даному алгоритмі кожний рядок стискається незалежно, враховуючи з якої точки він починається. Використовується у TIFF форматі. Дає такі коефіцієнти стиснення: найкращий 8, середній 3/2, найгірший 1. [5]

### 1.4.7 KWE (Keyword Encoding)

Алгоритми групи KWE являються алгоритмами кодування за ключовими словами. В основу їх роботи покладено кодування лексичних одиниць вихідного документу групами байтів фіксованої довжини. В якості лексичних одиниць виступають символьні послідовності, в яких символи повторюються. Ці послідовності кодуються ланцюжками кодів меншої довжини. В результаті кодування отримується таблиця, яка додається до результуючого коду і є словником. Для кодування англомовних текстів використовують двобайтне кодування слів, пари при цьому називаються токенами.

Найбільш відомими алгоритмами з цієї групи є LZ та LZW. В свою чергу алгоритм LZ має два головні варіанти реалізації — LZ77 та LZ78. Алгоритм LZ77 коротко називають варіантом «пам’ятай недавню історію», а LZ78 «пам’ятай те, що використовував», завдяки головним ідеям, які вони використовують.

Під час роботи алгоритму LZ створюється словник, в якому записано слово та його порядковий номер. На початку роботи алгоритм працює з незаповненим словником, в якому міститься лише один закодований NULL-рядок . Коли відбувається зчитування символу вхідної послідовності даних, то він додається до відповідного рядку. Так продовжується до моменту, коли поточний рядок відповідає фразі зі словника. Коли поточний рядок являє собою останнє співпадіння зі словником та новий зчитаний символ повідомлення, кодер видає код, що складається з індексу співпадіння та слідуючого за ним символу, що порушує співпадіння рядків. Нова фраза, яка складається з індексу співпадіння та слідуючого за ним символу, записується у словник. Ця фраза може бути використана для побудови більш довгої фрази, якщо вона зустрічається у повідомленні ще раз. Це сприяє підвищенню стиснення інформації.

Визначимо простий варіант словникового алгоритму, для його реалізації потрібно:

1. Ініціалізувати словник зі всіма символами, що зустрічаються у вхідному рядку.
2. Знайти в словнику найдовшу послідовність «С», що співпадає з початком кодує мого повідомлення.
3. Видати код знайденої послідовності і видалити її з початку кодує мого повідомлення.
4. Якщо кінець повідомлення не досягнуто, то врахувати наступний символ «с» і додати «Сс» до словника, перейти до другого кроку. Якщо досягнуто — вихід з алгоритму.

Розглянемо приклад: нехай задано рядок у вигляді такої послідовності символів «ЛЕЛЕКАЗЛЕЛЕКОЮЛЕТІЛА».

Для початку необхідно ініціалізувати словник для заданого вхідного повідомлення (таблиця 1.8):

Таблиця 1.8 — Ініціалізація словника

| Символ | Код символу |
|---|---|
| Л | 00001 |
| Е | 00010 |
| К | 00011 |
| А | 00100 |
| З | 00101 |
| О | 00110 |
| Ю | 00111 |
| Т | 01000 |
| І | 01010 |

Далі в процесі стиснення словник доповнюється послідовностями, що зустрічаються у повідомленні. Даний процес представлено у таблиці 1.9:

Таблиця 1.9 — Заповнення словника

| Послідовність, що кодується | Код | Послідовність у словник | Код |
|---|---|---|---|
| Л | 00001 | ЛЕ | 01011 |
| Е | 00010 | ЕЛ | 01100 |
| ЛЕ | 01011 | ЛЕЛ | 01101 |
| К | 00011 | ЕК | 01110 |
| А | 00100 | КА | 01111 |
| З | 00101 | АЗ | 10000 |
| ЛЕ | 01011 | ЗЛ | 10001 |
| ЛЕК | 01101 | ЛЕЛЕ | 10010 |
| О | 00110 | КО | 10011 |
| Ю | 00111 | ОЮ | 10100 |
| ЛЕ | 01011 | ЮЛ | 10101 |
| Т | 01000 | ЛЕТ | 10110 |
| І | 01010 | ТІ | 10111 |
| Л | 00001 | ІЛ | 11000 |
| А | 00100 | ЛА | 11001 |

При описі алгоритму не описувалася ситуація повного заповнення словника. В залежності від варіанту використання алгоритму можлива різна поведінка, а саме повне та часткове очищення словника, зупинка заповнення словника. Кожен з таких підходів має свої плюси та мінуси.

При очищенні словника може виникнути ситуація, коли видаляються часті послідовності.

При зупинці заповнення можливе зберігання в словнику послідовностей, які зустрічаються на початку рядку, проте які не вживаються надалі. Тому більшість реалізацій при заповненні словника відстежуються також ступінь стиснення і при його зниженні відбувається перебудова словника.

Існує велика кількість похідних LZ алгоритму, які відрізняються різними властивостями. Наприклад, методом пошуку повторюваних ланцюжків. Один з простих варіантів LZ алгоритму реалізується так: вважається, що у вхідному потоці [лічильник, зміщення відносно поточної позиції] або [лічильник] байт, які пропускаються та значення цих байт. При декодуванні для пари [лічильник, зміщення] копіюється [лічильник] байт вихідного масиву на [зміщення] байт, а [лічильник] пропущених байт копіюється у вихідний масив з вхідного потоку. Даний алгоритм є несиметричним.

Ефективність алгоритму залежить від довжини документу, адже до результату додається також словник.

### 1.5 Порівняння базових алгоритмів стиснення

Враховуючи описані у попередніх підрозділах особливості алгоритмів стиснення, складемо їх порівняльну таблицю 1.10.

В таблиці алгоритми порівнюються за такими характеристиками: клас зображень, для яких його бажано використовувати, симетричність алгоритму, особливості роботи, а також графічні формати, в яких для стиснення застосовується певний алгоритм.

Для порівняння розглядаються три популярні алгоритми, які вважаються базовими для систем стиснення, а саме: RLE, алгоритм Хаффмана та LZW [5].

Таблиця 1.10 — Порівняння алгоритмів

| Назва алгоритму | RLE | Алгоритм Хаффмана з фіксованою таблицею | LZW |
|---|---|---|---|
| Клас зображень | Зображення з невеликою кількістю кольорів та великими областями з одним кольором. Ділова графіка. | Використовується як етап компресії в більш складних системах. | Орієнтований на 8-бітні зображення, що побудовані на комп'ютері. |
| Симетричність | Приблизно одиниця | Два (потребує два проходи масивом стиснених даних) | Майже симетричний |
| Характерні особливості | Швидкий, не потребує додаткової пам'яті при роботі. | Простий у реалізації, не збільшує розмір вихідних даних в найгіршому випадку | Універсальний, ситуація збільшення розміру майже не виникає |
| Формати | PCX, TIFF, BMP | TIFF | TIFF, GIF |

Проаналізувавши ці дані можемо визначити, що час кодування та декодування для алгоритмів RLE та LZW приблизно рівний. Алгоритм Хаффмана та LZW не призводять до ситуацій збільшення розміру файлу після кодування. Проте алгоритм Хаффмана майже не вживається окремо, його застосовують у складних системах стиснення.

### 1.6 Висновки

В даному розділі було визначено головні поняття теми дипломного проекту для подальшої роботи з ними та досліджень. Було здійснено огляд наявних рішень, основних графічних форматів та базових алгоритмів стиснення зображень. Також було визначено основні сфери використання наведених рішень та визначено проблематику роботи. Таким чином, далі досліджуватиметься метод оптимізації GIF формату, як підвиду відеоінформації.

# 2 ПОСТАНОВКА ТА АЛГОРИТМ РОЗВ'ЯЗКУ ЗАДАЧІ

## 2.1 Визначення мети та цілі роботи, задачі

Метою даної роботи є створення нового способу покращення GIF формату, а також нового прийому оптимізації анімації. Спосіб являє собою обробку початкових кадрів анімації для зменшення їхніх розмірів. Обробка зображень проходить у два етапи: алгоритмом JPEG, а після цього алгоритмом LZW у формат GIF.

Задачі дипломної роботи включають в себе ознайомлення з основними поняттями обраної теми, огляд популярних форматів зображень та алгоритмів стиснення. Дослідження класифікації алгоритмів та розгляд принципів їх роботи. Детальний огляд GIF та JPEG форматів зображень, дослідження та порівняння роботи алгоритмів LZW та JPEG. Реалізація програмного додатку для швидкого перетворення зображення у GIF та JPEG формати та їх модифікації. Дослідження попереднього стиснення файлу та його впливу на розмір кадру, як способу оптимізації GIF анімації.

Цілями роботи є зниження розміру GIF анімації і як наслідок, зменшення часу завантаження веб-сторінок, які їх використовують, та скорочення зайнятого ними місця на фізичних носіях і хмарних сховищах. А також для збільшення швидкості, об'єму опрацьованої інформації.

Вихідними даними до роботи являється інформація про будову GIF файлів та методів їх стиснення. Програмний додаток реалізований мовою Java з використанням бібліотеки javax.imageio.

## 2.2 Алгоритм LZW

В попередньому розділі було розглянуто усі основні алгоритми стиснення графічної інформації, серед яких був словниковий метод. Алгоритм LZW являється модифікацією розробленого раніше алгоритму LZ78. Свою назву отримав від перших літер з прізвищ вчених, якими був розроблений — Лемпель,

Зів та Велч. В LZW алгоритмі стиснення відбувається за рахунок однакових ланцюжків байт. Це універсальний алгоритм стиснення без втрат. Він являється однопрохідним, що означає, що при кодуванні та декодуванні зображення не потрібно попередньо аналізувати вхідну інформацію.

Алгоритм заснований на простій ідеї. Загалом можна сказати, що LZW заміняє рядки символів деякими кодами. Це відбувається без попереднього аналізу вхідного тексту. Замість цього при додаванні нового рядку символів, переглядається таблиця рядків. Стиснення відбувається за рахунок заміни рядку символів кодом. Алгоритм LZW генерує коди будь-якої довжини, проте головною умовою є те, що вони мають містити більше біт, ніж одиничний символ. Перші 256 кодів відповідають стандартному наборові символів. Наступні коди відповідають оброблюваним алгоритмом рядкам.

Для процесу стиснення необхідно перетворити вхідний потік даних у потік кодів, використовуючи таблицю рядків, яка створюється в процесі кодування або декодування. Коли стискаються байтові данні, початковим станом таблиці є наявність 258 рядків, що містять послідовні коди від 0 до 255, а також спеціальні дев’яти бітові коди 256 та 257. Далі в процесі кодування чи декодування будуть створюватися нові рядки таблиці, що міститимуть ланцюжки байт, що кодуються, та посилання для організації таких ланцюжків. По ходу кодування, алгоритм переглядає текст посимвольно і зберігає кожний новий, унікальний двосимвольний рядок у таблицю у вигляді пари [код, символ], де код містить посилання на відповідний перший символ. Після того як новий рядок збережено у таблицю, на вихід передається код першого символу. Коли на вході зчитується наступний символ, для нього по таблиці знаходиться рядок максимальної довжини, що вже зустрічався. Після цього у таблиці зберігається код цього рядку з наступним символом на виході. На вихід подається код цього рядку, а наступний символ використовується в якості початку наступного рядку. Таблиця може сягати максимальної довжини у 4096 рядків.

Алгоритм LZW реалізований у форматах зображень GIF, TIFF, TGA. Його коефіцієнти компресії залежать від різних параметрів, таких як вибір

зображення, і складають приблизно: 1000 для найкращого випадку, 4 для середнього та 5/7 для найгіршого. [5] При чому стиснення в 1000 разів можливо досягти лише на моно-кольорових зображеннях, розмір яких кратний 7 Мб.

Алгоритм найкраще використовувати для таких класів зображень: восьмибітні зображення, що побудовані на комп'ютері. LZW стискає графіку за рахунок однакових підланцюжків у потоці.

Майже симетричній, проте при умові оптимальної реалізації операції пошуку рядку в таблиці.

До переваг алгоритму відносяться: висока швидкість роботи при кодуванні та декодуванні, невисокі вимоги до пам'яті, проста апаратна реалізація, відсутність дефектів після кодування у вхідному графічному файлі, відсутність потреби розрахунків ймовірностей появи символів.

Недоліками LZW є: низький ступінь стиснення в порівнянні з системою кодування у два проходи, алгоритм не проводить аналіз вхідних даних, через що не може вважатися оптимальним.

### 2.2.1 Робота алгоритму

Символи вхідного потоку послідовно зчитуються і відбувається перевірка, чи наявний в таблиці рядків такий рядок. Якщо рядок наявний, то зчитується наступний символ, а якщо рядок відсутній, то в потік вноситься код попереднього знайденого рядку, рядок вноситься у таблицю і пошук починається спочатку.

Нові рядки формують таблицю поетапно, тобто індекс рядку можна вважати його кодом. Для алгоритму декодування на вході потрібно отримати закодований текст, а таблиця перетворень може бути створена безпосередньо використовуючи закодований текст. Алгоритм генерує однозначний код для декодування за рахунок того, що кожного разу, коли генерується новий код, новий рядок додається до таблиці рядків. LZW перевіряє чи рядок уже відомий і якщо так, виводить відповідний рядок без генерування нового. Таким чином забезпечується, що кожний рядок зберігається в єдиному екземплярі і має свій

унікальний номер. Відповідно, при декодуванні при отриманні нового коду генерується новий рядок, при отриманні уже відомого рядку, він обирається зі словнику.

Виділимо такі кроки роботи алгоритму:

1. Ініціалізація словника усіма можливими одно символьними фразами. Ініціалізація вхідної фрази p першим символом повідомлення.
2. Зчитування символу «C» з повідомлення, що кодується.
3. Якщо досягнуто кінця повідомлення, то видається код для вхідної фрази p.
4. Якщо це не кінець повідомлення, то відбувається присвоєння вхідній фразі значення p(C) — символу, що кодується — та перейти до кроку 2. Інакше, видається код p, значення p(C) додається до словника. Вхідній фразі присвоюється значення «C» і відбувається перехід до кроку 2.
5. Кінець роботи.

Розглянемо приклад стиснення та декодування зображення.

Для початку відбувається створення початкового словника з одиничних символів. Так як у стандартному ASCII кодуванні є 256 різних символів, для їх правильного кодування, початковий розмір коду дорівнює 8 біт. Якщо нам відомо, яка кількість різних символів буде у вхідному файлі, то кількість біт можна зменшити. Для того щоб ініціалізувати таблицю встановлюється відповідність коду 0 символу з бітовим кодом 00000000, 1 — 00000001 і так далі до коду 255. Таким чином вказується, що коди від 0 до 255 є кореневими. Інших кореневих кодів у таблиці не буде.

Зі збільшенням розміру словника, розмір груп також буде рости, для того щоб врахувати нові елементи. Усі 8-бітні групи дають 256 можливих комбінацій біт, тому з появою 256 слова, алгоритм переходить до 9-бітних груп. Відповідно при появі 512 слова відбувається перехід до 10-бітних груп і так далі.

В прикладі розглянемо ситуацію, коли алгоритму відомо, що буде використовуватися лише 5 різних символів. Тому для їх зберігання використовується 3 біти для 8 різних комбінацій, що являється мінімальною кількістю біт для запам’ятовування цих символів.

Дана така вхідна послідовність: «тгтстгтмтгтстгтк». Виконаємо проходження усіх кроків алгоритму. Створена таблиця 2.1 результатів кодування:

Таблиця 2.1 — Таблиця для алгоритму LZW

| Поточний рядок | Поточний символ | Наступний символ | Вивід: код | Вивід: біти | Словник |
|---|---|---|---|---|---|
| Тг | Т | г | 0 | 000 | 5 – тг |
| Гт | Г | т | 1 | 001 | 6 – гт |
| Тс | Т | с | 0 | 000 | 7 – тс |
| Ст | С | т | 2 | 010 | 8 – ст |
| Тг | Т | г | - | - | - - |
| Тгт | Г | т | 5 | 101 | 9 – тгт |
| Тм | Т | м | 0 | 000 | 10 – тм |
| Мт | М | т | 3 | 011 | 11 – мт |
| Тг | Т | г | - | - | - - |
| Тгт | Г | т | - | - | - - |
| Тгтс | Т | с | 9 | 1001 | 12 – тгтс |
| Ст | С | т | - | - | - - |
| Стг | Т | г | 8 | 1000 | 13 – стг |
| Гт | Г | т | - | - | - - |
| Гтк | Т | к | 6 | 0110 | 14 – гтк |
| К | К | - | 4 | 0100 | - - |

Спочатку додаємо до пустого рядку «т» і перевіряємо чи наявний у таблиці рядок «т». Даний рядок присутній, адже при ініціалізації таблиці ми внесли туди рядки з одного символу.

Далі зчитуємо наступний символ з вхідного потоку «г» та перевіряємо чи наявний рядок «тг» у таблиці. Перевірка дала результат, що такий рядок у таблиці відсутній. Тому додаємо до таблиці рядок «тг» з кодом 5. В потік передаємо 0.

Продовжуємо перевірку:

- Рядок «гт» відсутній. Додаємо в таблицю «гт» з кодом 6. Передаємо у потік 1;
- Рядок «тс» відсутній. Додаємо в таблицю «тс» з кодом 7. Передаємо у потік 0;
- Рядок «ст» відсутній. Додаємо в таблицю «ст» з кодом 8. Передаємо у потік 2;
- Рядок «тг» наявний. Рядок «тгт» відсутній. Додаємо в таблицю «тгт» з кодом 9. Передаємо у потік 5;
- Рядок «тм» відсутній. Додаємо в таблицю «тм» з кодом 10. Передаємо у потік 0;
- Рядок «мт» відсутній. Додаємо в таблицю «мт» з кодом 11. Передаємо у потік 3;
- Рядок «тгт» наявний. Рядок «тгтс» відсутній. Додаємо в таблицю «тгтс» з кодом 12. Передаємо у потік 9;
- Рядок «ст» наявний. Рядок «стг» відсутній. Додаємо в таблицю «стг» з кодом 13. Передаємо у потік 8;
- Рядок «гт» наявний. Рядок «гтк» відсутній. Додаємо в таблицю «гтк» з кодом 14. Передаємо у потік 6;
- Рядок «к» останній, після нього йде кінець повідомлення. Тому виводимо у потік 4.

В результаті кодування отримано повідомлення «0 1 0 2 5 0 3 9 8 6 4». Якщо порівняти початковий розмір повідомлення і закодованого, то визначаємо, що розмір закодованого займає менше на 11 біт.

### 2.3 Алгоритм JPEG

Алгоритм JPEG був розроблений і стандартизований у 1991 році групою експертів в області фотографії Joint Photographic Expert Group (за цією назвою таку ім’я отримав алгоритм) для стиснення 24-бітних зображень. Алгоритм

використовує багато методів та технік, але можна сказати що він заснований на дискретно-косинусному перетворенні (ДКП), яке застосовується до матриці зображення для отримання деякої нової матриці коефіцієнтів.

ДКП розкладає зображення за амплітудами деяких частот. Тому після перетворення отримується матриця, в якій багато коефіцієнтів близькі, або дорівнюють нулю. Крім того алгоритм використовує особливості людського зору для більшої апроксимації коефіцієнтів без помітної втрати якості. Для цього використовується також квантування коефіцієнтів.

Отже, JPEG являється достатньо потужним алгоритмом. Він надзвичайно популярний і фактично вважається стандартом для фотографічних зображень. Алгоритм оперує областями зображення 8x8, на яких плавно змінюються яскравість та колір. В наслідок цього після розкладання матриці такої області в подвійний ряд по косинусам, лише перші коефіцієнти виявляються значимими. Таким чином, стиснення в JPEG відбувається за рахунок плавності зміни кольорів в зображенні.

Головними перевагами алгоритму є:

- Можливість задання ступеню стиснення;
- Вихідне кольорове зображення може мати 24 біти на точку.

До основних недоліків JPEG відносять:

- Зменшення якості зображення при збільшенні ступеню стиснення. Зображення розпадається на окремі квадрати, розміром 8x8. Це викликано тим, що відбуваються великі втрати у низьких частотах при квантуванні. Відновити ж вхідні дані неможливо.
- Поява дефектів, таких як ореоли Гіббса (ореоли біля границь з різким переходом кольорів).

Також однією з особливостей JPEG є поява горизонтальних та вертикальних смуг при друці зображення, хоча вони не будуть видні на екрані. Тому в поліграфії не рекомендується використання JPEG.

Властивості алгоритму JPEG:

- Ступінь стиснення від 2 до 200, може встановлюватися користувачем;
- Рекомендується використання для 24-бітних зображень, зображень в різних градаціях сірого, без різких переходів кольорів. Для фотографій;
- Алгоритм має симетричність близьку до одиниці.

Важливою особливістю JPEG є те, що він розроблявся саме як метод стиснення, а не стандарту представлення зображень. [7] Основними цілями JPEG являються: отримання високого коефіцієнту стиснення, можливість налаштування параметрів, задовільні результати для будь-яких типів зображень з плавними тонами та переходами, задовільна складність алгоритму для реалізації апаратно.

### 2.3.1 Робота алгоритму

Процес стиснення в алгоритмі JPEG досить складний, бо використовує різні техніки обробки зображення та покроково їх застосовує. Розглянемо етапи роботи алгоритму [2]:

**Перехід з RGB в YCrCb. Дискретизація**

Переведення зображення з простору RGB в колірний простір YCrCb (розглядалися в першому розділі роботи). Перетворення можливі для використання через особливості людського зору, який менш чутливий до кольору, ніж до яскравості, тому з’являється можливість архівувати масиви для компонент Cr та Cb з великими втратами. Завдяки цьому досягається вищий ступінь стиснення.

Перехід з RGB в YCrCb відбувається з використанням матриці переходу, формула (2.1):

$$\begin{vmatrix} Y \\ Cr \\ Cb \end{vmatrix} = \begin{vmatrix} 0{,}299 & 0{,}587 & 0{,}114 \\ 0{,}5 & -0{,}4187 & -0.0813 \\ 0{,}1687 & -0.3313 & 0{,}5 \end{vmatrix} * \begin{vmatrix} R \\ G \\ B \end{vmatrix} + \begin{vmatrix} 0 \\ 128 \\ 128 \end{vmatrix} \quad (2.1)$$

Зворотне перетворення відбувається завдяки множенню на зворотну матрицю, формула (2.2):

$$\begin{vmatrix} R \\ G \\ B \end{vmatrix} = \begin{vmatrix} 1 & 0 & 1{,}402 \\ 1 & -0{,}34414 & -0{,}71414 \\ 1 & 1{,}772 & 0 \end{vmatrix} * \left( \begin{vmatrix} Y \\ Cr \\ Cb \end{vmatrix} - \begin{vmatrix} 0 \\ 128 \\ 128 \end{vmatrix} \right) \quad (2.2)$$

**Дискретно-косинусне перетворення (ДКП)**

Вхідне зображення на даному етапі розбивається на матриці розміром 8x8, які називаються одиницями даних. Далі з них формуються три робочі матриці ДКП, по вісім біт окремо для кожної з компонент. Для того щоб отримати високі ступені стиснення, на даному етапі з вхідної матриці розміром 16x16 отримують одну робочу матрицю ДКП. При цьому втрачається майже 3/4 інформації про складові кольору зображення, але на результуюче RGB зображення ці зміни впливають не дуже сильно.

Спрощено ДКП, для n=8, можна представити так — формула (2.3):

$$Y[u,v] = \frac{1}{4} \sum_{i=0}^{7} \sum_{j=0}^{7} C(i,v) * C(j,v) * y[i,j], \quad (2.3)$$

де $C(i,u) = A(u) * \cos\left(\frac{(2*i+1)*u*pi}{2*pi}\right)$, $A(u) = \begin{cases} \frac{1}{\sqrt{2}}, u = 0 \\ 1, u \neq 0 \end{cases}$

ДКП застосовується до кожної з робочих матриць. При цьому в результаті отримується матриця, в якій коефіцієнти у верхньому лівому кутку відповідають низькочастотній складовій зображення, а в правому нижньому — високочастотній.

Поняття частоти випливає з розгляду зображення як двомірного сигналу: плавна зміна кольору відповідає низькочастотній складовій, а різка — високочастотній.

**Квантування**

В процесі стиснення зображення на даному етапі відкидаються коефіцієнти ДКП, які вносять несуттєвий вплив на відновлення зображення, яке буде близьке до оригінального. Квантування являється основним процесом під час якого відбуваються втрати даних у алгоритмі JPEG.

Квантування відбувається за рахунок поділу поелементно робочої матриці на матрицю квантування. Для кожної з компонент YCrCb задається власна матриця квантування (МК), що визначена формулою (2.4):

$$Yq[u,v] = IntegerRound\left(\frac{Y[u,v]}{q[u,v]}\right) \quad (2.4)$$

Задаючи матриці квантування з більшими коефіцієнтами, отримується більше нулів, що забезпечує більший ступінь стиснення. В стандарті JPEG є також рекомендовані МК. Матриці для більшого або меншого ступеню стиснення отримуються шляхом множення вхідної матриці на деяке число (коефіцієнт гамма). Якщо коефіцієнт буде завеликим, зображення може розпастися на квадрати 8x8 через великі втрати у низьких частотах. При цьому через втрати у високих частотах з’являються ореоли Гіббса.

**Зиг-заг сканування**

Наступним етапом проходить зиг-заг сканування, за допомогою якого матриця 8x8 переводиться у 64-елементний вектор. Елементи обираються по черзі, за індексами — (0, 0), (0, 1), (1, 0), (2, 0) і так далі. Завдяки скануванню на початку вектора отримуються коефіцієнти матриці, що відповідають низьким частотам зображення, а вкінці — високим.

Зображення схеми сканування матриці, отриманої після квантування, знаходиться на рисунку 2.1:

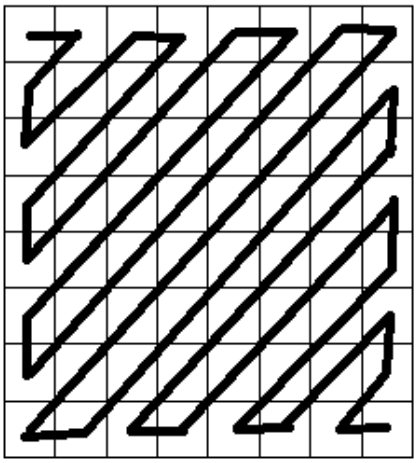

Рисунок 2.1 — Схема зиг-заг сканування

**Групове кодування**

Отриманий вектор далі згортається алгоритмом групового кодування, наприклад RLE. При цьому отримуються пари [зміщення, число], де «зміщенню» відповідає лічильник нулів, які пропускаються, а «число» визначає значення, яке необхідно внести в наступну комірку.

**Алгоритм Хаффмана**

На цій стадії отримані пари кодуються алгоритмом Хаффмана з фіксованою таблицею. Замість алгоритму Хаффмана може застосовуватися також арифметичне кодування.

Повну блок-схему основних етапів роботи алгоритму JPEG зображено на кресленику з шифром ІА52.070БАК.006 Д4.

## 2.4 GIF формат

В основу формату GIF покладено алгоритм стиснення LZW, який був створений Лемпелем, Зівом та Велчем як нова покращена реалізація вже відомого на той час алгоритму LZ78. Перша специфікація формату була

представлена у 1987 році компанією Compuserve і з того часу він почав активно використовуватися у практично всіх сферах, а особливої популярності набув в інтернеті. GIF формат є форматом растрових зображень. Як відомо, растрові зображення являють собою матрицю (або сітку) з пікселів, які містять у собі інформацію про колір. Основними характеристиками GIF-формату є:

- підтримка до 256 кольорів, для опису яких застосовується від 1 до 8 бітів на піксель;
- кольори палітри мають глибину в 24 біта на піксель;
- зберігання в одному файлі набору зображень (так звані «гіфки» — GIF-анімації);
- для анімацій доступна велика кількість повторів, від 1 до 65535, також можлива нескінченна кількість повторів;
- однією з особливостей формату є підтримка прозорості (доступна карта прозорості);
- розмір зображення може бути від 1x1 до 65535x65535 пікселів;
- також можливе додавання необмеженої кількості текстових коментарів.

Незважаючи на те, що даний формат підтримує досить малу кількість кольорів, його великою перевагою являється саме алгоритм стиснення. Адже алгоритм LZW являється алгоритмом стиснення без втрат, тому якість зображення після обробки зображення не страждає. Також така скорочена кількість кольорів у палітрі і як наслідок скорочення кількості графічних даних, дозволяє зберегти досить малий розмір файлу, що забезпечує швидке завантаження графіки. А це є дуже важливим показником, тому що це означає, що GIF формат буде доцільно використовувати при повільних мережевих з’єднаннях.

Всі ці особливості формату дозволили йому завоювати велику популярність в мережі. Даний формат отримав всезагальне визнання. Проте через появу проблем з патентом на LZW алгоритм та деякі юридичні питання

його використання різко скоротилося. Це також вплинуло на подальший розвиток і розробку формату. Відомо, можна використовувати також інші алгоритми стиснення для реалізації даного формату. Але в рамках дипломного проекту буде розглянуто лише GIF формат на основі LZW алгоритму.

Попри появу нових форматів, які виправляють недоліки GIF, він все ще активно використовується. Формат широко представлений у таких сферах як: реклама, дозвілля, мистецтво, а також спілкування між людьми та соціальних мережах. Розглянемо його вплив на кожну з цих сфер.

В рекламні сфері GIF використовується для зображень з чіткими контурами та суцільними кольорами, без переходів — при створенні логотипів. Але найактивніше застосовується GIF анімація. Дизайнери створюють рекламні банери, в яких записана послідовність графічних зображень, що зберігає вільний простір на сторінці, а також звертає на себе увагу користувачів. Такі банери швидко і лаконічно подають інформацію, створюють відчуття руху і можуть продемонструвати або пояснити роботу продукту. GIF анімація вимагає на розробку набагато менше затрат в ресурсах часу та грошах, на відміну від відео. Крім того, GIF майже не впливає на швидкість завантаження сторінки, що є надзвичайно важливим показником для сайту. Також даний формат підтримують усі браузери, без необхідності додаткових налаштувань та встановлень плагінів.

Найбільшої популярності GIF формат здобув у сфері дозвілля, спілкування між людьми та соціальних мережах. Згідно з даними відомої соціальної мережі Tumblr, його користувачі щодня викладають на своїх сторінках в загальному 23 мільйони GIF анімацій. [8] Тому даний сервіс навіть додав функцію створення анімацій у своєму додатку. Інші соціальні мережі також переповнені GIF анімаціями. У їхні додатки вбудовуються функції пошуку «гіфок». Користувачі використовують їх замість слів та навіть цілих речень. Анімації прекрасно передають емоції, відсилки на популярні історії, жарти, фільми. Це є дуже важливим інструментом для спілкування. GIF анімації також підтримують усі популярні месенджери. Створюються спеціальні сервіси для пошуку потрібних анімацій, що впливає на поширення формату.

Завдяки поширенню в соціальних мережах, GIF анімації використовують у сфері дозвілля та навіть мистецтві. З’явилися такі нові напрямки мистецтва, як Cinemagraph та GIF-арт.

Можна помітити, що своєю популярністю GIF формат завдячує саме своїй особливості зберігання в одному файлі набору зображень та створення анімацій. Далі детальніше розглянемо що являє собою GIF анімація та пропозиції щодо їх покращення.

### 2.4.1 GIF анімація

Як було зазначено раніше, формат GIF не втрачає свою популярність саме завдяки своїй особливості, що вирізняє його серед інших форматів — здатності зберігати в одному файлі набір із зображень. Тож визначимо точніше, що таке GIF анімація. Це послідовність, що складається з декількох статичних кадрів та інформації про те, скільки кожен з кадрів буде показаний на екрані — затримки. Затримкою називають індивідуальний час показу кадру, який вимірюється в сотих долях секунди. При чому затримка для різних кадрів може бути різною, що дозволяє скоротити кількість використаних зображень (для відео при «застиганні» моменту будуть використовуватися однакові кадри; в GIF анімації можна скористатися одним кадром, встановивши для нього більшу затримку).

В анімації можна встановлювати кількість повторів, а також можна її зациклити — в цьому випадку після закінчення відображення останнього кадру завжди починатиметься показ першого кадру.

Завдяки анімаціям можна також обходити обмеження на 256 кольорів палітри. Для цього між кадрами, які містять різні палітри, потрібно встановлювати нульову затримку, в результаті чого кадри будуть накладатися.

По своїй суті GIF анімації — це не відео-файл, а зображення. Проте вони є альтернативою коротким відео, на їх розробку буде витрачено менше ресурсів і вони мають менший розмір. Для створення анімацій можна використовувати багато спеціальних програм  та графічних редакторів, таких як Ulead Gif Animator, Photoshop.

Для перегляду GIF анімацій можна використовувати будь-яку програму-переглядач. Проте в програмах, де не підтримується анімація, буде відображатися лише перший кадр з файлу. GIF анімації підтримуються у всіх браузерах, що також грає на їх користь.

Популярні сайти, які використовують GIF [3]:

- Google.com
- Facebook.com
- Baidu.com
- Tmall.com
- Yahoo.com
- Amazon.com

Хоча поки що популярної альтернативи для GIF анімації немає, це не означає, що не можна покращувати саму GIF анімацію. Для цього використовують оптимізацію — набір прийомів та технік, з метою отримання скорочення об’єму файлу. Основними прийомами є:

- скорочення кількості кольорів;
- скорочення кількості кадрів;
- використання оптимальної роздільної здатності;
- обрізання нерухомих частин картинки.

В даному дипломному проекті буде запропоновано новий прийом оптимізації, а саме обробка початкових кадрів для зменшення їхніх розмірів.

### 2.5 Висновки

У даному розділі було визначено мету та цілі роботи. Розглянуто роботу та особливості алгоритмів LZW та JPEG. Також було визначено особливості GIF формату та його властивості створювати анімації. Доведено актуальність проблеми, що розглядається та сфери використання рішення.

# 3 РОЗРОБЛЕНИЙ КОМПЛЕКСНИЙ АЛГОРИТМ

У даному проекті було розроблено програмний засіб на основі комплексного алгоритму стиснення. Його робота будується на поєднанні алгоритмів LZW та JPEG, а саме поетапного їх застосування. Розроблений алгоритм пропонується до використання для оптимізації зображень GIF формату. На першому етапі алгоритму відбувається перекодування зображення методом JPEG. Після цього відбувається повторне стиснення інформації LZW алгоритмом. Таким чином досягаються кращі результати стиснення при тому, що втрати точності відображення не відбуваються. Для кращого розуміння роботи запропонованого алгоритму у даному розділі розглянуто будову формату GIF та JFIF, а також розглянуто роботу складових частин запропонованого алгоритму — LZW та JPEG.

## 3.1 Структура та внутрішня будова файлу GIF формату

Формат GIF зберігає в собі багато байтові цілі числа з молодшим байтом на першому місці, у прямому порядку байтів. Зчитування рядків бітів відбувається у напрямку від наймолодшого біта до найстаршого.

Файли GIF формату мають свою складну будову. Вони складаються з фіксованого заголовку — області на початку, певної кількості блоків в залежності від модифікації,  та кінцевим блоком — завершувачем зображення [9]. Розглянемо детальніше кожну частину файлу.

**Заголовок**

Кожний файл формату GIF починається із заголовка. Заголовок допомагає програмі-переглядачу визначати, що файл саме GIF формату та якої версії формат. В залежності від версії формату, інформація в заголовку буде різною. Наприклад у форматі GIF87a, заголовок міститиме виключно опис зображення, а у форматі GIF89a він включатиме також блоки розширень. Заголовок GIF зазвичай складається з двох полів, які займають 3 байти — «Сигнатура»

(значення цього поля буде «GIF») та «Версія» (значення цього поля буде «87a» або «89a», в залежності від версії).

**Логічний дескриптор екрану**

Цей блок також називають «описом». Він визначає логічну область екрану, в якій будуть відображатися окремі зображення з GIF файлу. Логічним екраном називають область реального екрану комп'ютера, куди виводяться усі зображення даного GIF файлу. Вони можуть мати різний розмір та займати різне положення на логічному екрані. Дескриптором вказуються розміри області та колір фону, який обирається з глобальної таблиці кольорів. Для того щоб визначити в якому місці логічного екрану має бути розміщено зображення, використовують дескриптори окремих зображень. Деструктор екрану має досить складну структуру та включає в себе 7 байт інформації. На рисунку 3.1 нижче, приведено будову деструктора.

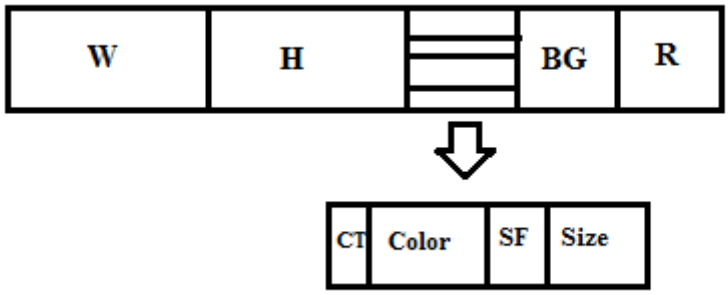


Рисунок 3.1 — Будова деструктора

Перші два поля W та H займають по два байти і відповідають за ширину та висоту логічного екрану у пікселях — тобто розмір області виведення зображень. Ті зображення, які мають розмір, більший за логічний екран, повинні бути обрізані до його розміру.

Поле BG — номер кольору фону, займає один байт і являється вказівником кольору в глобальній таблиці кольорів. Області фону, де відсутні зображення, заливаються кольором з глобальної палітри файлу. Проте якщо в найпершому

розширення управління графікою увімкнена прозорість, то колір вважається прозорим.

Поле R називають полем «Пропорції розмірів пікселя». Дане поле займає 1 байт. Наразі дане поле не використовується, адже передбачає, що піксель екрану може бути не квадратним, що було можливим лише у старих комп'ютерах. У версії 87a дане поле є зарезервованим і замість значень приймає нулі. Проте якщо значення поля є ненульовим, це означає, що ширина і висота пікселях не співпадають.

У байті з упакованими полями та прапорцями складна будова.

Так, поле-прапорець CT визначає наявність глобальної палітри кольорів. Займає 7 біт і встановлюється, коли у файлі використовується глобальна таблиця кольорів. Це означає, що відразу після закінчення дескриптора слідуватиме початок глобальної палітри.

Поле Color відповідає за роздільну здатність кольорів початкового зображення. Завдяки значенню цього поля, можна визначити як було створено GIF файл — з повноколірного зображення чи індексованого. Якщо з повноколірного, то значення поля буде 7. Якщо з індексованого, то значення залежатиме від глибини кольору індексованого зображення.

Наступне поле SF являється прапорцем сортування палітри, або таблиці кольорів. Займає 3 біти і встановлюється, коли кольори в глобальній таблиці кольорів відсортовані в порядку важливості. Тобто вказує чи сортована палітра за значимістю кольорів. Значимість кольору визначається площею зображення, яку він займає відносно інших кольорів. В версії 87a дане поле зарезервовано і встановлено в 0.

Поле Size пов'язано з полем CT і визначає розмір палітри та кількість кольорів зображення. Якщо прапорець CT скинуто, то в поле записуються нулі. Поле займає від 0 до 2 бітів.

Число записів в глобальній палітрі вираховується за формулою (3.1):

$$2^{(N+1)} \quad (3.1)$$

де N — значення поля Size.

**Глобальна палітра**

Якщо у дескрипторі був встановлений прапорець CT, то відразу після закінчення дескриптора починається блок глобальної палітри. Вона діє на всіх зображеннях, у яких немає локальної палітри. Глобальна палітра являє собою масив певних структур, а саме три поля: Червоне R, Зелене G та Синє B. Всі вони займають один байт, і в них записані значення червоної, зеленої та синьої компоненти відповідно. Число записів у масиві визначається у дескрипторі, а саме у полі Size.

Цей блок необхідний GIF файлу, адже зображення, які в ньому зберігаються, є індексованими. Це означає, що вони складаються з номерів кольорів, а самі кольори знаходяться в палітрі. Палітра ж складається з так званих тріад, які складаються з червоного, зеленого та синього кольорів.

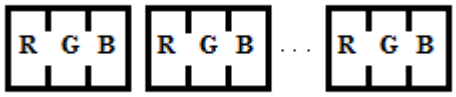


Рисунок 3.2 — Будова глобальної палітри

Якщо у файлі відсутні і глобальна, і локальна палітри, то програма-переглядач може використати системну палітру. Але це крайній випадок і рекомендується, щоб було задано хоча б перші два кольори — білий та чорний, що забезпечить виведення зображення.

**Дескриптор зображення**

Даний блок напряму пов’язаний з наступним, після нього — графічним. За відсутності дескриптору зображення, саме зображення не буде виведено. Тому його сприймають як одне ціле з блоком зображення і називають «заголовком блоку зображення». Структуру даного блоку зображено на рисунку 3.3:

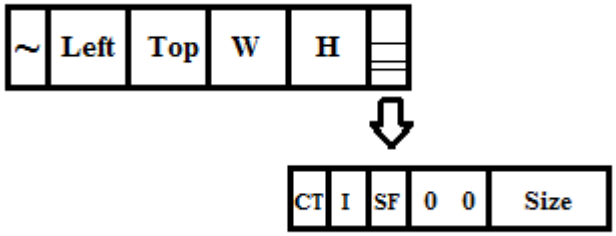


Рисунок 3.3 — Будова дескриптора зображення

Перше поле являється роздільником зображень.

Поля Width W та Height H — ширина та висота зображення у пікселях. Вони займають по два байти.

Left Position та Top Position також займають по два байти та визначають положення зображення на логічному екрані. Наприклад, поле Left Position вказує на зміщення картинки вліво всередині логічного екрану.

У складному байті з упакованими полями, поле «00» зарезервовано і має містити нулі.

Поле CT визначає наявність локальної палітри для зображення. Його також називають «прапорцем локальної таблиці кольорів». CT займає сім біт і якщо прапорець встановлено, це означає, що зображення використовує локальну палітру. Вона починається відразу після деструктора зображення.

Поле I визначає розгортку зображення при виводі на екран. Дана особливість дозволяє швидше отримати перші обриси зображення при завантаженні файлу в мережі.

Прапорець SF відповідає за сортування палітри. Він займає п’ять біт. Якщо прапорець встановлено, це означає, що кольори в локальній палітрі відсортовані в порядку їх значимості. У версії 87a, дане поле зарезервовано і встановлене в нуль.

Поле Size визначає розмір локальної палітри та кількість кольорів зображення за таким самим принципом, як у деструкторі логічного екрану. Поле займає від нуля до двох біт.

**Локальна палітра**

Якщо прапорець CT не був встановлений у нуль, тоді відразу після дескриптора йде локальна палітра. Вона діє лише на наступний пілся неї графічний блок.

**Графічний блок (блок даних)**

У цій частині файлу знаходиться безпосередньо зображення, стиснене методом LZW. Кількість картинок у файлі необмежена. Це означає, що можливе створення GIF анімації завдяки послідовним блокам графіки.

Даний блок також має окремі поля: MC, S та зображення, розбите на субблоки по 255 байт. Поле MC визначає початковий розмір LZW-коду. Розмір дорівнює глибині кольору картинки. Але виключення становить зображення з двох кольорів, тоді MC дорівнює 2. Поле S визначає розмір субблоку даних, який має бути рівний 255 байт. Лише для останнього субблоку розмір може бути від 1 до 255 байт. Ланцюжок з таких наборів полів закінчується блоком даних з нульовими байтами даних. Поле «00» є термінатором блоку.

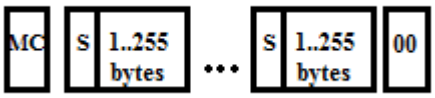


Рисунок 3.4 — Будова графічного блоку

Крім цих основних блоків, можуть бути також додаткові, так звані блоки розширення. Вони додаються в залежності від необхідності та версії формату. Розглянемо також блок розширення управління графікою, який використовується для створення анімації.

**Розширення управління графікою**

Даний блок визначає як необхідно відображати кожне наступне зображення в файлі. Розширення зазвичай використовують для визначення порядку відображення кадрів в GIF анімації. Цей блок був введений у версії 89a.

Розширення управління графікою має будову, зображену на рисунку 3.5 нижче:

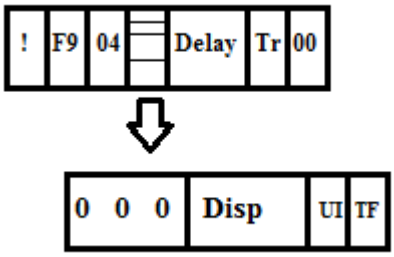


Рисунок 3.5 — Будова розширення управління графікою

У поле «!» записується ідентифікатор блоку розширення, воно є текстовою константою. Поле «F9» — ярлик розширення управління графікою, поле з записом «04» визначає розмір субблоку даних (4 байти). Ці поля відповідають шістнадцятирічним константам.

Поле Delay визначає час затримки, який зображення відображатися на екрані. Також затримку визначають як проміжок часу, протягом якого декодер чекає перед продовженням обробки наступних даних. Час затримки може встановлюватися від 0,01 с до 655 с. Поле займає 2 байти.

Поле Tr відповідає за використання вказівника прозорого кольору. Номер відповідає номеру кольору з локальної або глобальної палітри.

Поле з записом «00» являється термінатором блоку та шістнадцятирічною константою.

У байті у упакованими полями, поля «000» відповідають за зарезервовані поля, в яких мають стояти нулі.

Поле Disp  називають «методом вивільнення пам’яті». Займає від двох до чотирьох байтів. Вказує, що має робити декодер після відображення картинки (спосіб заміни зображення після показу). Розрізняють такі способи:

- 0 — відсутність дій;
- 1 — залишити зображення на тому ж місці;
- 2 — відновити колір фону;

- З — відновити дані, які були перед відображенням зображення.

Прапорець UI — прапорець користувацьких вхідних даних. За умови, що прапорець встановлено, програма-переглядач повинна очікувати вводу даних від користувача перед відображенням наступного зображення. Майже не використовується.

Прапорець TF відповідає за прозорість. Якщо встановлений — зображення відображається з прозорим фоном.

### 3.2 Структура та будова JPEG файлу

JPEG файл також представляється як потік бітів. Порядок розміщення бітів виглядає так: цілі числа зберігаються у файлі зі старшим бітом на першому місці, що означає вони зберігаються у зворотному порядку. При цьому бітові рядки кодуються, починаючи з старших бітів, всередині байтів.

Файли, які відповідають стандарту JPEG мають такі розширення: .jpg, .jpeg, .jfif, .jpe. Сам стандарт JPEG не вказує програмі-переглядачу, як їй необхідно відкривати зображення. JPEG не надає інформацію про те, як відображаються кольори, лише задає як зберігають значення компонентів графіки. Тому була створена специфікація JFIF. Даний формат визначає зображення, яке передається. Проте JPEG файл та JFIF являються синонімами.

Файли формату JPEG складаються з маркерів. Маркери використовують для розбиття потоку даних JPEG файлу на певні структури компонентів. Кожен з маркерів займає два байти, з яких перший байт завжди заповнений значенням $FF_{16}$. В другий байт записується код, який зберігає та вказує який тип маркера використовується.

Загалом маркери мають приблизно однакову структуру, а саме складаються з ідентифікатора, довжини та даних. Другий байт, який вказує на тип маркера і являється ідентифікатором. Довжина (або лічильник довжини) займає два байти, які містять довжину даних маркера, проте не включають

довжину самого маркера. Дані маркера — це дані, які мають бути оброблені в залежності від типу маркера.

Зазвичай розрізняють два типи маркерів JPEG:

- Одиничні;
- Маркери з даними.

Одиничні маркери, крім своїх двох байтів, не містять більше ніякої інформації. А після маркерів з даними відразу слідує значення в два байти, яке вказує на число байтів даних, що містяться в маркері. Основні одиничні маркери: $RST_0$-$RST_7$, SOI, EOI. Маркери з даними: $APP_n$, COM, DHT, DRI, DQT, $SOF_n$.

Формат JPEG не має чіткої структури щодо порядку розташування маркерів. Проте існують декілька обов'язкових загальних правил:

1. Файл JPEG має починатися маркером SOI та закінчуватися маркером EOI.
2. Якщо дані з одного маркера необхідні для обробки іншого маркера, то перший маркер повинен розташовуватися перед другим.
3. Деякі маркери ($RST_N$) можуть зустрічатися всередині стиснених даних, але не зустрічаються в маркерах.
4. Стиснені дані компонентів не можуть зустрічатися всередині маркерів. Вони розташовуються після маркеру SOS.

Відповідно до четвертого правила, виділимо окремо стиснені дані. Вони являються єдиною частиною JPEG файлу, що знаходиться поза межами маркерів.

Як було зазначено вище, дані записуються відразу після маркеру SOS, в якому немає інформації про довжину. Тому для того щоб знайти кінець стиснених даних, програмі необхідно просканувати дані до наступного маркеру. При цьому програма пропускає маркер $RST_N$, так як він може міститися всередині стиснених даних.

Розглянемо детальніше деякі типи маркерів.

**Маркери $APP_n$**

Цей тип маркерів містить у собі специфічні дані програми. $APP_n$ маркери використовуються для програм переглядачів та графічних редакторів для збереження додаткової інформації, окрім інформації стандарту JPEG. В залежності від програми-переглядачу, формат маркера може змінюватися.

Сюди може записуватися інформація, яка не властива формату, але необхідна програмі. Маркер $APP_n$ може розташовуватися в будь-якій частині JPEG файлу. Значення n може набувати від 0 до 15.

Особливістю маркера є те, що програми, які створюють даний маркер зберігають в його початок своє ім'я — для того щоб попередити конфлікти з іншими програмами. Ім'я програми має закінчуватися нулем. В такому випадку наступна програма, що оброблятиме файл, буде перевіряти не тільки ідентифікатор маркеру, а й ім'я програми, що його створила.

**Маркери COM**

Маркер, що відповідає за коментарі: він зберігає у собі рядки коментарів до файлу. При чому коментарі мають бути записані саме у COM маркер, тому що декодер буде ігнорувати цю інформацію. Даний маркер може бути розташований у будь-якій частині файлу.

**Маркери DHT**

Назва DHT маркеру розшифровується як «визначник таблиці Хафмана». Даний маркер визначає таблиці Хафмана, які ідентифікуються типом та числом. Маркер має складну структуру і складається з трьох полів, які описують таблицю. Перший байт маркера визначає ідентифікатор таблиці. Наступні 16 байтів об'єднуються в масив, що складається з однобайтних цілих чисел без знаку і задають число кодів Хафмана для кожної можливої довжини коду. Сума з 16 довжин і є числом значень в таблиці. Для того, щоб визначити число таблиць Хафмана, його записують в поле довжини.

**Маркер DRI**

Маркер, що відповідає за визначення інтервалу перезапуску всередині стиснених даних. Значення поля довжиною два байти, яке слідує за маркером, завжди рівне 4.

Даний маркер містить лише одне поле даних, значення, яке визначає інтервал перезапуску. Поле займає два байти і якщо в нього записані нулі, то це означає, що маркери перезапуску не використовуються. Якщо значення відмінні від нуля, то вони можуть використовуватися для повторної активізації маркерів перезапуску.

Маркер DRI може знаходитися у будь-якій частині файлу і визначати інтервал перезапуску та змінювати його, коли це потрібно.

Ці маркери необхідні для нейтралізації помилок, які можуть виникнути при скануванні. Коли декодер знаходить пошкоджені дані зі скану, він використовує ідентифікатор маркера перезапуску і інтервал перезапуску, таким чином визначаючи місце, з якого має відновитися декодування.

**Маркери DQT**

Маркер «визначення таблиць квантування» визначає таблиці квантування, які використовуються у зображенні. Він складається з поля довжини маркера та поля визначення таблиць.

Маркер може бути розміщений у будь-якій частині файлу, проте якщо скану необхідна таблиця дискретизації, то вона має бути попередньо визначена DQT маркером.

**Маркери EOI**

EOI маркер буквально називається «кінець зображення» і відмічає кінець зображення. Тому логічно, що даний маркер знаходиться вкінці JPEG файлу. Цей маркер може використовуватися лише один раз і після нього не можуть бути розташовані інші маркери.

**Маркери SOI**

Маркер початку зображення. Може розташовуватися лише один раз на початку файлу. SOI визначає початок зображення.

**Маркери $SOF_n$**

Відповідають за початок кадру та визначають кадр. Маркер має фіксований заголовок, поле довжини, список структур, які визначають компоненти кадру.

У заголовку вказується висота та ширина зображення у пікселях (поля займають по два байти), точність дискретизації в бітах (займає один байт), число компонентів зображення (один байт).

В залежності від компонентів, наступна частина маркера містить записи про ідентифікатор компоненту, частоту дискретизації по горизонталі та вертикалі, ідентифікатор таблиці квантування для компонента.

Маркер $SOF_n$ може бути записаний в файлі лише один раз, перед маркером SOS.

**Маркери SOS**

Маркери використовують для визначення початку сканування, а саме вказують на початок стиснених даних для стиснення в потоці. Самі стиснені дані скану записуються безпосередньо після маркеру.

Маркер складається з п’яти полів: лічильник компонентів, дескриптори компонентів скану, початок та кінець спектрального виділення, послідовне наближення.

Даний маркер з’являється у файлі після $SOF_n$ маркеру, при чому перед ним мають бути вказані маркери DHT та DQT.

**Маркери $RST_n$**

Значення n набуває від 0 до 7. Даний маркер використовують для маркування блоків кодованих стиснених даних скану. Особливістю маркера є те,

що він знаходиться тільки всередині стиснених даних. У нього відсутні поля довжини та даних.

Цей тип маркерів також можуть використовувати для нейтралізації помилок. Інтервал між маркерами $RST_n$ задається маркером DRI.

### 3.2.1 Формат JFIF

Як зазначалося вище, файлом JPEG фактично називають «файл JPEG у форматі JFIF». Даний стандарт визначає сигнатуру для ідентифікації JPEG файлів, простір кольорів, густоту пікселів та мініатюри зображень.

Уся додаткова інформація для JFIF зберігається у маркерах APP, тому даний формат повністю сумісний з JPEG. Єдиною умовою використання JFIF є те, що після маркеру SOI має бути маркер $APP_0$, який зберігає необхідну інформацію. Маркер $APP_0$ для цієї специфікації має відповідати певному формату. Він містить дев'ять полів:

- Ідентифікатор (5 байт) — рядок JFIF, що завершується нулем;
- Основний код версії (1 байт) — ідентифікаційний код версії файлу;
- Додатковий код версії (1 байт) — додатковий ідентифікаційний код версії;
- Одиниці виміру (1 байт) — одиниці виміру густоти пікселів по горизонталі та вертикалі;
- X-густота (2 байти) — густота пікселів по горизонталі;
- Y-густота (2 байти) — густота пікселів по вертикалі;
- Два необов'язкових поля ширини та висоти додаткової мініатюри зображення (займають по 1 байту);
- Мініатюра — додаткова мініатюра зображення.

Також JFIF може містити додатковий маркер $APP_0$ для вставки мініатюри.

Ще однією особливістю стандарту JFIF є те, що він використовує простір кольорів YCbCr. Дана модель складається з трьох компонентів: яскравості Y,

компонентів кольоровості — Cb (задає синій відтінок) та Cr (задає червоність). Особливості даної моделі були описані у першому розділі проекту.

Повну структуру JFIF файлу зображено на кресленику з шифром ІА52.070БАК.005 Д2.

### 3.3 Особливості реалізації алгоритму LZW

Алгоритм LZW, розглянутий у попередньому розділі використовується у форматі GIF, оптимізація якого є метою роботи. Тому в цьому розділі розглянемо детальніше LZW алгоритм та його місце у GIF.

LZW алгоритм має декілька особливостей своєї реалізації у формі стиснення зображень GIF формату. Першою особливістю є змінний розмір коду таблиці рядків, який не може перевищувати 12 біт, тобто не перевищувати числа 4095. Наступна особливість полягає у використанні двох спеціальних кодів: коду оновлення таблиці рядків та коду закінчення потоку символів.

На початку роботи алгоритму відбувається визначення кількості кольорів, які використовуються у зображенні. Так як GIF формат підтримує лише 256 кольори в палітрі, то будь-яке зображення перетворюється у 256 колірний простір. Також є обмеження на мінімальну кількість кольорів — 2. У тому випадку коли використовується два кольори, то початковий розмір кодів у таблиці дорівнює 3 біти. Коду 0 відповідає колір 0, коду 1 — 1. Коди 4 та 5 відповідають кодам очистки таблиці та коду. Якщо у зображенні використовують велику кількість кольорів, то розмір коду таблиці дорівнює числу біт N, які припадають на один піксель. Початковий розмір кодів в таблиці записується в заголовок GIF файлу.

Кодування пікселів зображення починається кодами розміром в біт. Далі з заповненням таблиці, значення кодів збільшуються.

Для алгоритму LZW використовують різні додаткові методи підвищення швидкості роботи алгоритму і збільшення швидкості стиснення даних. Зазвичай вони пов’язані зі складною організацією таблиці для зменшення часу пошуку по

ним, а також з виконанням перетворень над даними перед їх обробкою та внесенням до таблиці.

### 3.3.1 Загальна ідея алгоритму

Для розуміння наступних пояснень роботи LZW в GIF форматі, визначимо деякі основні поняття.

Символом в растрових зображеннях є індекс, що вказує на колір відповідного пікселя. Ланцюжком являється деяка послідовність символів. Префікс має майже те саме визначення, як і ланцюжок, проте вважається, що префікс знаходиться безпосередньо перед символом і може мати нульову довжину. Коренем вважається ланцюжок з одного символу.

Код — число, яке кодує ланцюжок і визначене певною кількістю біт. Елементом вважатимемо код та його ланцюжок.

Представлення LZW алгоритму у простій реалізації за допомогою псевдокоду представлено нижче:

```
РЯДОК = черговий символ з вхідного потоку
ПОКИ вхідний потік не пустий ВИКОНАТИ
    СИМВОЛ = черговий символ з вхідного потоку
    ЯКЩО РЯДОК+СИМВОЛ наявний в таблиці рядків ТОДІ
        РЯДОК = РЯДОК + СИМВОЛ
    ІНАКШЕ
        Вивести у вхідний потік код для РЯДОК
        Додати у таблицю рядків запис РЯДОК+СИМВОЛ
        РЯДОК = СИМВОЛ
    КІНЕЦЬ ЦИКЛУ ЯКЩО
КІНЕЦЬ ЦИКЛУ ЯКЩО
Вивести у вхідний потік код для РЯДОК
```

### 3.3.2 Використання LZW в GIF

Алгоритм LZW має свої особливості реалізації у форматі GIF. У частині заголовку GIF-файлу існує спеціальне поле, яке має назву «розмір коду». Але фактично містить у собі розмір кореня. Фактичний розмір кодів стиснення змінюється в процесі кодування та декодування, назвемо цей розмір «розміром стиснення».

Як прийнято в алгоритмі LZW початкова таблиця містить у собі коди для усіх коренів, проте до її верхньої частини також додаються два спеціальні коди. Поле «розмір коду» зазвичай містить значення, рівне числу бітів на піксель. Позначимо це поле P. Якщо число бітів на піксель рівне 1, то P дорівнює 2, так як корені займатимуть комірки 0 та 1, а спеціальні коди комірки 4 та 5. В будь-якому іншому випадку P рівне числу бітів на піксель, корені займають комірки від 0 до $(2^P-1)$, а спеціальні коди $2^P$ та $2^P+1$ відповідно. Отже, значення спеціальних кодів залежить від мінімального розміру коду, що слідує за заголовком зображення. [10]

Початковий розмір стиснення дорівнює P+1 біт на код. Якщо ведеться кодування, то виводиться спочатку коди довжиною P+1, якщо ведеться декодування, то обираються P+1 біт з потоку кадрів. Що стосується спеціальних кодів: CC код очищення $2^P$, EOI кінець інформації $2^P+1$. Код CC повідомляє компресору про необхідність ініціалізувати таблицю рядків і скинути розмір стиснення до P+1. Код EOI означає, що кодів більше немає і це кінець повідомлення. Починати додавання елементів до таблиці рядків слід з CC+2 рядку. При кодуванні необхідно вивести код CC як перший код, а далі, коли досягається код 4095, так як GIF не дозволяє, щоб розміри стиснення були більші за 12 біт.

Насправді змінний розмір стиснення не дуже реалізувати. Якщо кодування починається з розміру стиснення в P+1 біт, то коли виведеться код $(2^P-1)$, необхідно збільшити розмір стиснення на один біт. Тоді наступний код буде на

один біт довшим. Проте якщо досягається код 4095, необхідно вивести код CC і почати спочатку.

### 3.4 Особливості реалізації алгоритму JPEG

У попередньому розділі було розглянуто загальну схему роботи алгоритму стиснення зображень JPEG. Далі розглянемо детальніше деякі етапи кодування.

**Дискретно-косинусне перетворення**

ДКП відіграє дуже важливу роль у роботі алгоритму JPEG. На даному етапі відбувається перетворення коефіцієнтів косинус них функцій зі збільшенням частот. При чому кожне з вихідних значень перетворюється у суму косинусів. У форматі JPEG ДКП оперує над даними, які згруповані у блоки розміром 8x8. І завдяки врахування кореляції між пік селями по горизонталі та вертикалі, можемо отримати кращі результати. Тому першим кроком у стисненні одиниці даних є виконання двомірного перетворення ДКП, яке описується для квадратної матриці розміром N такою формулою (3.2) (для JPEG значення N дорівнює 8):

$$T[i,j] = \\ = c(i,j)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} V[y,x]\cos\frac{(2y+1)i*pi}{2N}\cos\frac{(2y+1)j*pi}{2N} \quad (3.2)$$

де $c(i,j) = \frac{2}{N}, i$ та $j \neq 0; c(i,j) = \frac{1}{N}, i$ та $j = 0$.

Двомірне ДКП виконується перемноженням матриць і виражається таким чином, формула (3.3):

$$T = MVM^T, \quad (3.3)$$

де V — матриця одиниці даних розміром 8x8, M — матриця ДКП.

Далі для дискретних компонент Y у вигляді блоків 8x8, розраховуються коефіцієнти двомірного ДКП, які зберігаються у маркерах DQT та SOS JPEG файлу.

Головною ідеєю, яка лягає в основу ДКП являється відкидання коефіцієнтів більш високого порядку, так як вони, як правило, вносять до зображення меншу кількість інформації, і при цьому зображення залишається близьким до оригіналу. Ці коефіцієнти розташовані у верхньому лівому кутку матриці. Проте це стосується фотографічних зображень з плавними переходами. Для рисунків, де скачки кольорів досить різкі, будуть відбуватися значні втрати у якості.

**Квантування**

Пошук та відкидання коефіцієнтів, що вносять мінімальний внесок до формування зображення, відбувається на етапі квантування. Для виконання квантування коефіцієнтів ДКП необхідно розділити їх на інше значення і округлити, формула (3.4):

$$\text{Квантоване значення} = \text{Округлення}\left(\frac{\text{Коефіцієнт}}{\text{Значення кванту}}\right) \quad (3.4)$$

Для того, щоб визначити значення квантів для зображення, у JPEG використовується масив з 64 елементів, який називається таблицею квантування. Дана таблиця визначається у маркері DQT. Допускається використання декількох таблиць квантування, для того, щоб усі компоненти, що квантуються, не використовували однакові значення. Кожне значення таблиці квантування використовується для квантування відповідного коефіцієнту ДКП.

Стандарт JPEG пропонує декілька таблиць квантування, які були перевірені і показують добрі результати, проте програмою дозволяється використання будь-яких значень квантування. В результаті отримаємо таблицю, в якій буде багато нульових коефіцієнтів.

### Зиг-заг сканування

Для того, щоб згрупувати максимальну кількість квантованих нульових коефіцієнтів для створення довших серій нульових значень, коефіцієнти ДКП в одиницях кодують з використанням зиг-заг шляху. Таким чином всі нульові коефіцієнти будуть знаходитися вкінці ланцюжка з елементів.

## 3.5 Висновки

У даному розділі було детальніше досліджено роботу та особливості алгоритмів LZW та JPEG. Стиснення алгоритмом JPEG спирається на особливості людського зору та відбувається за рахунок огрубіння дрібних деталей зображення, зменшення плавності зміни кольорів. Стиснення алгоритмом LZW відбувається за рахунок наявності однакових ланцюжків байт та представлення їх у більш компактному вигляді. Тобто розглянуті алгоритми застосовують різні підходи та впливають на різні частини зображення і зберігання інформації.

Отже, можемо запропонувати застосовувати їх комплексно, в одній системі стиснення, для досягнення кращих результатів стиснення для формату GIF. Перевірка практичних результатів застосування такої системи буде представлена в наступному розділі.

# 4 РЕАЛІЗАЦІЯ

## 4.1 Етапи виконання роботи та пропозиція

Дослідження та виконання роботи проходило поетапно, слідуючи початковому плану виконання проекту. Відбувалося дослідження основних понять заданої тематики — відеоінформації, графічної інформації, форматів зображень, алгоритмів стиснень. Було визначено сфери застосування та огляд наявних рішень. Детальніше розглянуто формат GIF та GIF анімацію.

В результаті проведених робіт з дослідження проблематики сфери, було визначено необхідність оптимізації розміру файлів GIF формату, а точніше анімації.

Тому пропозицією даного дипломного проекту являється система з двоетапного стиснення зображення заснована на розробленому комплексному алгоритмі. Під час першого етапу відбувається обробка та стиснення зображення алгоритмом JPEG. Другим етапом являється стиснення отриманого зображення алгоритмом LZW та перетворення зображення у GIF формат.

Теоретична підстава для розробки такої системи була розглянута у попередніх розділах. В даному розділі представлені та проаналізовані результати застосування такого підходу до стиснення графічної інформації.

## 4. 2 Програмний додаток для перевірки роботи алгоритму

Для роботи системи стиснення комплексним алгоритмом було створено програмний додаток мовою Java.

Створена програма запускається через командний рядок. Далі обираємо нестиснених файл для обробки та коефіцієнт стиснення для алгоритму JPEG. Оброблені зображення зберігаються у директорії з кодом програми.

На рисунку 4.1 зображено частину процесу роботи розробленого додатку у командному рядку:

```
Converting source image to JPEG, no compression.
Converting source image to JPEG with compression
Enter compression quality (0-100):
20
Converting source image to GIF
Converting JPEG image to GIF
Converting compressed JPEG image to GIF
Images were converted succesfully.
```

Рисунок 4.1 — Результати роботи програми у командному рядку

Лістинг програмного коду знаходиться у додатку Б.

### 4.2.1. Пакет javax.imageio

При розробці програмного додатку використовувався пакет javax.imageio. Він являється основним пакетом для Java Image I/O API. Пакет допомагає читати та записувати файли зображень. Також завдяки ньому користувач може змінювати коефіцієнти стиснення зображень при їх збереженні. Бібліотекою Image I/O підтримуються формати BMP, GIF, PNG, JPEG. [11]

Для базового використання бібліотеки вводу-виводу, необхідні статичні методи класу ImageIO. Наприклад, метод read застосовують для читання зображення. Для запису існує метод write, при чому він може мати три форми, в залежності від особливостей використання у програмі.

Як було зазначено раніше, при записі зображення можна керувати різними метаданими, наприклад ступенем стиснення.

Для цього необхідно використовувати деякі інші класи та пакети, а саме ImageWriteParam (клас, що описує як має бути закодований потік), та працювати з провайдером writer. Щоб отримати параметри запису, які встановлені за замовчуванням для ImageWriter (абстрактний суперклас для кодування та запису зображень) користуються методом getDefaultWriteParam, що повертає об’єкт ImageWriteParam. Для зміни ступеня стиснення необхідно вказати об’єкту, що ступінь буде встановлений явно:

ImageWriteParam param = writer. getDefaultWriteParam();

param. setCompressionMode(ImageWriteParam.MODE_EXPLICIT);

Далі встановлюється необхідне значення за допомогою методу setCompressionQuality.

Оскільки параметри введення налаштовуються, то для представлення зображення, що записується необхідно користуватися об’єктом класу IIOImage (простий контейнерний клас для агрегації зображень та метаданих представлення об’єкта, що пов’язані із зображенням).

### 4.3 Реалізація LZW. Псевдокод

Теоретичні відомості та характеристики алгоритму LZW були розглянуті у попередніх розділах. Розглянемо частину однієї з можливих реалізацій даного алгоритму.

Для цього використовуватимемо такі змінні:

- array_pic[] (відповідає за масив пікселів — індексованих кольорів);
- current_position (визначає поточне положення у масиві);
- symbol (визначає символ з масиву);
- string_of_symbol (рядок символів);
- code (відповідає за готовий код).

Також визначимо деякі функції:

- ToAddString() — додає новий рядок до словника;
- ToSearch() — повертає номер знайденого рядку зі словника або якщо рядок не був знайдений, повертає (-1);
- ToWrite() — записує згенерований код у вихідний масив.

Нижче на рисунку представлено блок-схему, яка пояснює роботу програми:

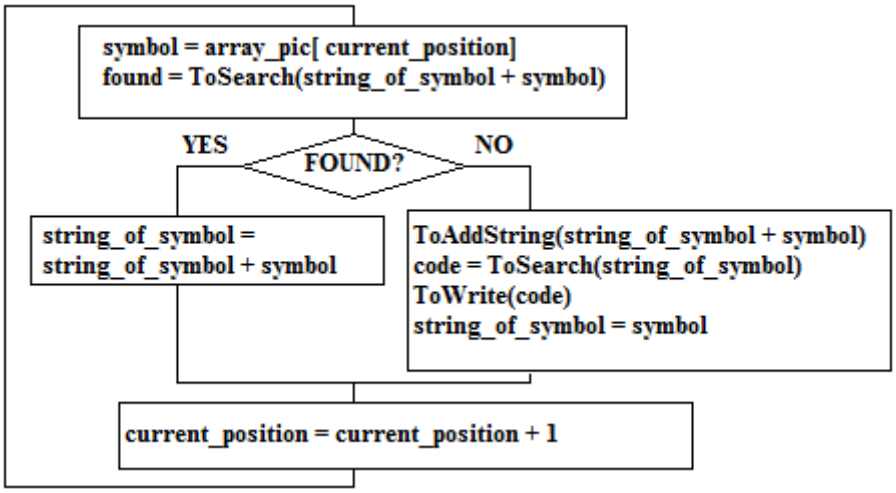


Рисунок 4.2 — Блок-схема частини алгоритму LZW

Стиснення за даною схемою буде відбуватися до тих пір, доки у потоці знаходяться необроблені символи. Вкінці до LZW-потоку додається завершальний код EOI. Перед цим кодом додається останній значущий код з адресою останнього рядку, що співпав зі словником (таблицею рядків).

### 4.4 Реалізація JPEG

Для розробки запропонованої системи стиснення використовуються вже відомі реалізації алгоритмів. Нижче запропоновано деякі варіанти кодування складових частин алгоритму JPEG.

Лістинг коду для створення дерева Хаффмана [12]:

```
template<class T>
  void huffman(MinHeap<TreeNode<T>*> heap, int n)
  {
   for(int i=0;i<n-1;i++)
   {
    TreeNode<T> *first = heap.pop();
    TreeNode<T> *second = heap.pop();
    TreeNode<T> *bt = new BinaryTreeNode<T>(first, second, first.data,
second.data);
```

```
            heap.push(bt);
          }
        }
```

Один з прикладів реалізації алгоритму RLE [13]:

```
#include <stdio.h>
#include <string.h>
int main()
{
   int cnt;
   char smb;
   char *code = new char [80];
   char *encode = new char [80];
   char *str = new char [80];
   scanf("%s", code);
   strcpy(encode, "");
   smb = code[0];
   cnt = 0;
   for (int i = 0; i <= strlen(code); i++) {
      if (code[i]==smb) {
         cnt++;
      }
      else {
         sprintf(str, "%d", cnt);
         strcat(encode, str);
         sprintf(str, "%c", smb);
         strcat(encode, str);
         smb = code[i];
         cnt = 1;
      }
   }
```

```
            printf("%s\n", encode);
            return 0;
        }
```

Дискретно-косинусне перетворення може бути визначене у програмному коді таким чином [2]:

```
 void DCT (unsigned int NN, double input [], double output[])
{
    double cc = 1/sqrt(NN);
    for (unsigned int ii =0; ii<NN; ++ii)
    {
        output [ii] = 0;
        for (unsigned int jj = 0; jj<NN; ++ii)
        {
            output[ii] += cc*(inpit [jj])*cos((2*jj+1)*ii*M_PI/2.0/NN);
        }
        cc = sqrt (2.0/NN);
    }
    return;
}
```

Запропоновані приклади реалізацій не являються єдиними можливими. Використовуючи теоретичні знання про алгоритми та кроки їх виконання, можливе створення інших реалізацій, які можуть бути ефективнішими чи швидшими. Як було зазначено, у даній роботі використовувалися реалізації алгоритмів у програмному пакеті мови Java javax.imageio.

### 4.5 Результати роботи програми

Завдяки розробленому програмному додатку та програмам перегляду зображень, було проведено перевірку результатів роботи системи стиснення у два етапи (комплексним алгоритмом). Для цього розглянуто набір зображень різних класів та з різними характеристиками, щоб визначити ефективність застосування алгоритму для різних зображень. Нижче представлено таблицю з отриманими даними розмірів зображення в залежності від застосованого алгоритму стиснення:

Таблиця 4.1 — Таблиця результатів застосування комплексного алгоритму

| | | Назва зображення | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Rock | Ray | Hill | Cat | Bird | Boy | Pic | Tx |
| | BMP | 4165 | 1407 | 258 | 770 | 255 | 386 | 10 | 4165 |
| | GIF | 447 | 109 | 106 | 561 | 4 | 263 | 10 | 423 |
| | JPEG | 149 | 50 | 44 | 92 | 17 | 77 | 29 | 190 |
| | JPEG20 | 56 | 21 | 16 | 27 | 8 | 24 | 16 | 74 |
| | JPEG+GIF | 404 | 97 | 251 | 370 | 23 | 198 | 43 | 439 |
| | JPEG20+GIF | 343 | 85 | 213 | 202 | 23 | 152 | 1 | 381 |
| | K(GIF) | 9.3 | 12.9 | 2.4 | 1.3 | 63.7 | 1.4 | 1 | 9.8 |
| | K(JPEG) | 27.9 | 28.1 | 5.8 | 8.3 | 15 | 5 | 0.3 | 21.9 |
| | K(JPEG20) | 74.3 | 67 | 16 | 28.5 | 31.8 | 16 | 0.6 | 56.2 |
| | K(JPEG+GIF) | 10.1 | 14.5 | 1.1 | 2.1 | 11.1 | 1.9 | 0.2 | 9.5 |
| | K(JPEG20+GIF) | 12.2 | 16.6 | 1.2 | 3.8 | 11.1 | 2.6 | - | 10.9 |

У таблиці навпроти назв форматів, у які було перетворене початкове зображення, записано розмір отриманого файлу у кілобайтах. Напроти коефіцієнтів стиснення, відповідно стоять визначені значення, за формулою (4.1):

$$\text{Коефіцієнт стиснення} = \frac{\text{Розмір файлу } .bmp}{\text{Розмір обробленого файлу}} \qquad (4.1)$$

Проаналізувавши отримані дані в таблиці, можемо відзначити, що алгоритм дає неоднозначні результати. Це залежить від класу зображень до яких його застосовано. Так, для чорно-білих зображень третього класу та зображень першого і другого класу застосування даного алгоритму є неефективним, адже розмір зображення збільшується, у порівнянні зі звичайним GIF. Це пояснюється тим, що алгоритм JPEG дає гірші ступені стиснення для графічних та чорно-білих зображень, і майже не застосовується для їх обробки. Для інших

перевірених зображень, запропонований алгоритм показав кращі коефіцієнти стиснення, досягнувши числа 16,6 для зображення Ray.bmp.

Зображення, які було використано для перевірки роботи алгоритму знаходяться у додатку А.

Для прикладу розглянемо детальніше, як показувало себе нестиснене та необроблене зображення формату BMP — Rock.bmp. Початковий розмір вихідного зображення становить 4165 Кб. Даний приклад відноситься до третього класу зображень: фотореалістичне зображення з великою кількістю кольорів та плавних переходів кольорів. Для кращого дослідження зміни якості зображення після обробки та стиснення, в програмі-переглядачі використаємо інструмент ZOOM для масштабування зображення і встановимо значення 200. Таким чином будемо порівнювати таку частину зображення:

Рисунок 4.3 — Початкова частина зображення Rock.bmp з ZOOM 200

Спочатку виконуємо перетворення вихідного зображення з формату BMP в GIF. При цьому застосовується алгоритм стиснення LZW. Таким чином розмір зображення зменшився і становить 447 Кб, що майже в дев’ять разів менше початкового розміру.

Проте відбулися зміни в якості зображення. Алгоритм LZW являється алгоритмом стиснення без втрат, тому не призводить до погіршення якості. Але так як однією з особливостей GIF формату є підтримка 256 кольорів у палітрі,

відбулося скорочення кількості використовуваних відтінків. За рахунок цього отримане зображення відрізняється від початкового. Проте всі деталі зображення залишаються доступними для розпізнавання і якість є задовільною.

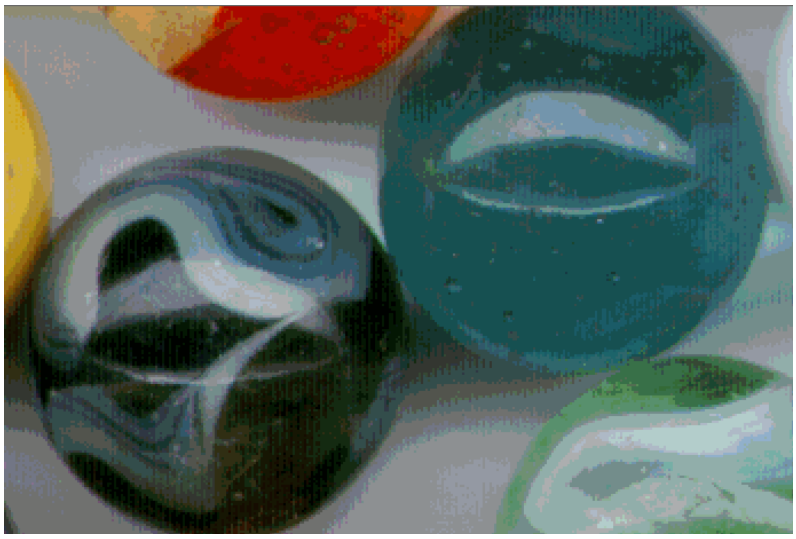

Рисунок 4.4 — Зображення Rock.bmp, оброблене методом GIF

Далі перетворимо початкове зображення BMP у графічний формат JPEG. При перетворенні застосовується алгоритм стиснення з втратами JPEG, тому звернемо увагу на зміни у якості зображення. Під час перетворення програмою пропонується задати параметри стиснення і коефіцієнт.

Розглянемо два випадки: з коефіцієнтом 100 та коефіцієнтом 20.

У першому випадку при стисненні втрати мають бути мінімальними. Зображення представлено на рисунку. Розмір файлу становить 149 Кб. Порівнюючи з початковим зображенням можна сказати, що зміни незначні. Проте помітно зменшилася яскравість та насиченість кольорів зображення, а також границі стали менш чіткими.

Рисунок 4.5 — Зображення Rock.bmp, оброблене методом JPEG

Другий випадок — JPEG з коефіцієнтом стиснення 20. Розмір файлу складає 56 Кб, що майже у 2,6 разів менше об’єму пам’яті, який займає попереднє зображення. Проте через встановлення такого коефіцієнту стиснення відбулися безповоротні зміни у якості зображення: зменшення насиченості та яскравості кольорів, розмиття границь кольорів, втрата деяких дрібних елементів графіки. Незважаючи на це, людський зір все може ще розрізнити дане зображення і відмітити його подібність до початкового. Але дане зображення краще не використовувати для друку та у сферах застосування, де необхідна чіткість та висока якість.

Рисунок 4.6 — Зображення Rock.bmp, оброблене методом JPEG 20

Наступне зображення, яке розглядається є результатом застосування запропонованої у проекті системи стиснення у два етапи.

Спочатку вхідне зображення формату BMP стискається та перетворюється у формат JPEG з коефіцієнтом стиснення 100. Після цього отримане зображення перетворюємо у GIF формат, використовуючи при цьому алгоритм стиснення LZW. Результатом стало зображення розміру 404 Кб.

Так як нас цікавить саме GIF формат, порівнюватимемо отримані результати з зображенням GIF. Розмір зображення зменшився у 1,1 раз, при цьому відбулися деякі незначні втрати якості. Майже непомітно зменшилася яскравість кольорів зображення та змінився порядок пікселів. Проте цього непомітно неозброєним оком і зображення майже неможливо відрізнити від звичайного GIF файлу.

Тому такий підхід до обробки зображення дає кращі результати, ніж використання лише перетворення у GIF формат.

Рисунок 4.7 — Зображення Rock.bmp, оброблене методом JPEG+GIF

Останнім для перевірки роботи двоетапної системи стиснення обираємо зображення, яке оброблялося алгоритмом JPEG з коефіцієнтом стиснення 20, а після цього алгоритмом LZW.

Отримали зображення у форматі GIF розміром 343 Кб. У порівнянні зі звичайним GIF файлом, об’єм пам’яті, яку займає зображення скоротився у 1,3 рази. Так як для створення GIF анімації використовується декілька кадрів-зображень, то результуючий виграш у розмірі буде значним. Порівняємо якість зображень: зменшення насиченості та яскравості кольорів, розмиття границь кольорів, втрата деяких дрібних елементів графіки. Поява незначних артефактів, зменшення плавності переходів кольорів.

Якість постраждала адже зміни відбулися на двох етапах. Але попри це, зображення все ще дуже подібне до початкового і людський зір може знехтувати даними проблемами.

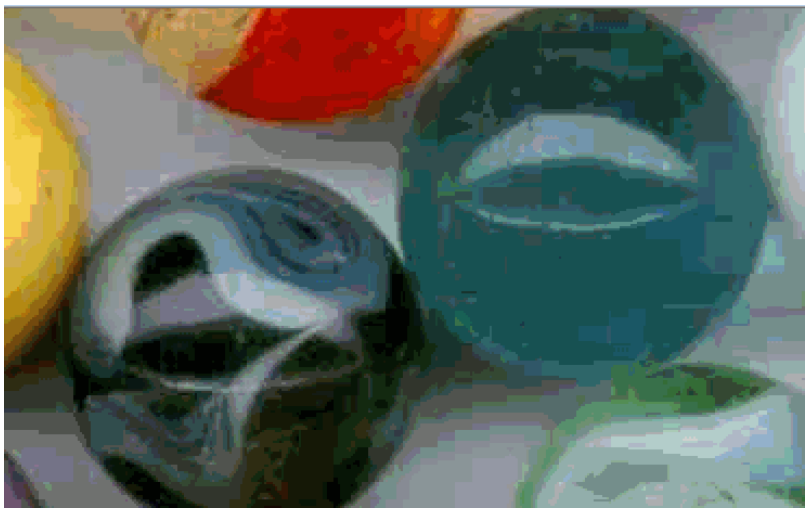

Рисунок 4.8 — Зображення Rock.bmp, оброблене методом JPEG20+GIF

Отже, ефективність даного підходу до стиснення доведена і двоетапне стиснення може бути запропоновано для залучення до інших систем стиснення або використане незалежно. З його допомогою можна зменшувати об’єм пам’яті, що має бути виділений для зберігання зображень. При цьому якість зображення незначно погіршується та не втрачається його інформаційна цінність.

Проте для отримання задовільних результатів необхідно враховувати клас і особливості зображення, яке буде оброблюватися.

### 4.6 Варіанти подальшого розвитку роботи

Зважаючи на те, що комплексний алгоритм показав свою ефективність для стиснення зображень формату GIF, що означає, що він може принести практичну цінність.

Даний алгоритм може бути застосований у сферах використання GIF формату, наприклад у рекламі, мистецтві, дозвіллі, комунікації людей та у соціальних мережах. Для детального ознайомлення з сферою застосування GIF формату можна звернутися до розділів 1 та 2 дипломного проекту.

Також алгоритм можна застосовувати для оптимізації GIF анімацій — завдяки попередній обробці кадрів та зменшення об’єму пам’яті, яку вони займають, зменшується загальний розмір анімації.

Завдяки розробці нових програмних реалізацій алгоритмів LZW та JPEG можна збільшувати їхню ефективність і як результат, збільшувати ефективність роботи комплексного алгоритму. Також комплексний алгоритм можна включати у інші існуючі системи стиснення для отримання кращих результатів.

### 4.7 Висновки

У даному розділі було описано етапи виконання роботи та пропозицію дипломного проекту. Також було описано програмний додаток, який заснований на роботі комплексного алгоритму. У розділі проведено огляд пакету мови Java завдяки якому реалізовано представлений комплексний алгоритм. Розглянуто етапи роботи алгоритму та представлено приклади реалізації його складових частин.

Проведено огляд результатів роботи додатку використовуючи набір різних зображень та проаналізовано ефективність запропонованого засобу стиснення. Розглянуто сфери можливого використання результатів проекту та варіанти подальшого розвитку роботи.

# ВИСНОВКИ

У даному дипломному проекті розроблено програмний засіб, що реалізує комплексний алгоритм стиснення для оптимізації інформації у форматі GIF.

Також у даній роботі було досліджено стиснення відеоінформації у розрізі її складових частин — кадрів, було вивчено різні формати зображень та їх алгоритми стиснення, а також визначено існуючі на сьогодні проблеми у сфері перекодування. Тому метою проекту було виділення задачі оптимізації перетворення зображень та пропозиції щодо її вирішення. Таким чином було запропоновано та розроблено комплексний алгоритм для застосування до графічного формату GIF.

Перш за все було визначено поняття відеоінформації, типи зображень та види графіки, виділено основні класи зображень. Все це необхідно було для подальшого розгляду графічних форматів та розуміння чому певні види зображень зберігаються в окремих форматах, тобто для визначення сфер застосування. Також було описано основні графічні формати та виділено випадки їх використання, визначено популярність форматів на ринку станом на сьогоднішній день. Для визначення актуальності проблеми, було проведено аналіз GIF формату та його порівняння зі звичайним відео.

Було розглянуто базові алгоритми стиснення та досліджено їх принципи роботи, проведено покрокове стиснення на прикладах рядків повідомлень. Для визначення переваг та недоліків алгоритмів, а також сфер їх застосування, також було створено таблицю з порівняльною характеристикою алгоритмів.

У другому розділі описано алгоритми LZW та JPEG. Було розглянуто роботу алгоритму LZW на прикладі та визначено основні етапи роботи складного алгоритму JPEG. Наведено інформацію про історію створення та використання у сферах життя формату GIF та GIF анімації. Таким чином було доведено актуальність розробки для застосування у сучасних реаліях.

Далі для обґрунтування напрямків досліджень було детально розглянуто внутрішню будову та структуру файлу GIF формату та будову файлу JFIF. Так як

основами розроблюваного алгоритму в два етапи стали алгоритми LZW та JPEG, було детально розглянуто особливості їх реалізації, загальні ідеї. Для алгоритму LZW, який уже застосовується для стиснення файлів GIF формату, було визначено особливості його використання у кодуванні формату.

Наступний розділ дипломного проекту був присвячений безпосередньо реалізації представленої пропозиції. Пропозицією у даній роботі являється система з двоетапного стиснення зображення. Під час першого етапу відбувається обробка та стиснення зображення алгоритмом JPEG. Другим етапом являється стиснення отриманого зображення алгоритмом LZW та перетворення зображення у GIF формат. Для розробки програмного додатку, який швидко виконує перекодування у різні графічні формати було використано мову Java та її пакет javax.imageio, в якому вже містяться вбудовані класи з реалізацією роботи необхідних алгоритмів. Також для ознайомлення було наведено приклади інших реалізацій етапів алгоритму JPEG та псевдокоду алгоритму LZW.

У цьому розділі також було представлено результати застосування розробленого алгоритму у вигляді результатів роботи програмного додатку. Для порівняння та проведення аналізу результатів використовувався набір зображень різних класів та характеристик. Таким чином було доведено ефективність алгоритму та визначено для яких класів зображень його доцільно застосовувати.

Після проведення аналізу отриманих результатів роботи було визначено сферу застосування даного алгоритму, а також розглянуто можливі варіанти подальшого розвитку роботи. Даний алгоритм може бути застосований у всіх сферах використання GIF формату, які були описані у перших розділах дипломного проекту. Також алгоритм можна застосовувати для оптимізації GIF анімацій, які користуються великою популярністю і являються важливою властивістю GIF формату. Для подальшого розвитку алгоритму його можна застосовувати у складних системах стиснення та покращувати його складові.

Отже, мету дипломного проекту було досягнуто та реалізовано поставлені задачі у вигляді продуктів програмного забезпечення.

# ПЕРЕЛІК ПОСИЛАНЬ


1. John Miano. Compessed image file formats — Addison-Wesley Professional, 1999 — 288 с
2. Ватолин Д. С. Алгоритмы сжатия изображений/ Методическое пособие — Москва: «Московский государственный университет им. М. В. Ломоносова», 1999 [Електронний ресурс] — Режим доступу: http://lib.ru/TECHBOOKS/ALGO/VATOLIN/algcomp.htm.
3. “Historical trends in the usage of image file formats for websites” [Електронний ресурс] — Режим доступу: https://w3techs.com/technologies/history_overview/image_format/all .
4. “Usage statistics of GIF for websites ” [Електронний ресурс] — Режим доступу: https://w3techs.com/technologies/details/im-gif/all/all .
5. Ватолин Д., Ратушняк А., Смирнов М., Юкин В. Методы сжатия данных. М.: ДИАЛОГ-МИФИ, 2013 — 381 с.
6. Тулякова М. С. «Методы и алгоритмы сжатия графической информации»/ научный руководитель кандидат тех. н., д. Б. А. Крылов/ Научно-технический вестник. Университет ИТМО/ 2016.
7. Крилов Є. В., Анікін В. К., Бугаєнко О. С. «Покращення характеристик алгоритму стиснення JPEG з метою підвищення швидкості завантаження сайтів»/ Адаптивні системи автоматичного управління: міжвідомчий науково-технічний збірник, № 1(26)/ 2015.
8. Mike Isaac . Article “For mobile messaging, GIFs prove to be worth at least a thousand words”/ The New York Times/ 03.08.2015 — Режим доступу: https://www.nytimes.com/2015/08/04/technology/gifs-go-beyond-emoji-to-express-thoughts-without-words.html?_r=0.
9. «Краткое описание формата GIF» [Електронний ресурс] — Режим доступу: http://home.onego.ru/~chiezo/gif.htm#gct.

10. Blackstock Steve. LZW and GIF explained [Електронний ресурс] — Режим доступу: https://www.eecis.udel.edu/~amer/CISC651/lzw.and.gif.explained.html.
11. Java documentation/ Class ImageIO [Електронний ресурс] — Режим доступу: https://docs.oracle.com/javase/8/docs/api/javax/imageio/ImageIO.html — Дата доступу: 09.05.2019.
12. Ellis Horowitz, Sartaj Sahni, Dinesh Mehta. Fundamentals of data structures in C++. First edition. W. H. Freeman, 1995 — 653 с.
13. «Реализации алгоритмов/ Кодирование длин серий» [Електронний ресурс] — Режим доступу: https://ru.wikibooks.org/wiki/Реализации_алгоритмов/Кодирование_длин_серий.

# ДОДАТОК А. ЗОБРАЖЕННЯ РІЗНИХ КЛАСІВ ДЛЯ ПЕРЕВІРКИ АЛГОРИТМУ

Для перевірки роботи розробленого додатку на основі комплексного алгоритму, а також для визначення ефективності алгоритму, було використано набір із зображень різних класів та з різними характеристиками. Використані зображення представлені на рисунках 1-8:

Рисунок 1 — Початкове зображення Rock.bmp


Рисунок 2 — Початкове зображення Ray.bmp

Рисунок 3 — Початкове зображення Hill.bmp

Рисунок 4 — Початкове зображення Cat.bmp

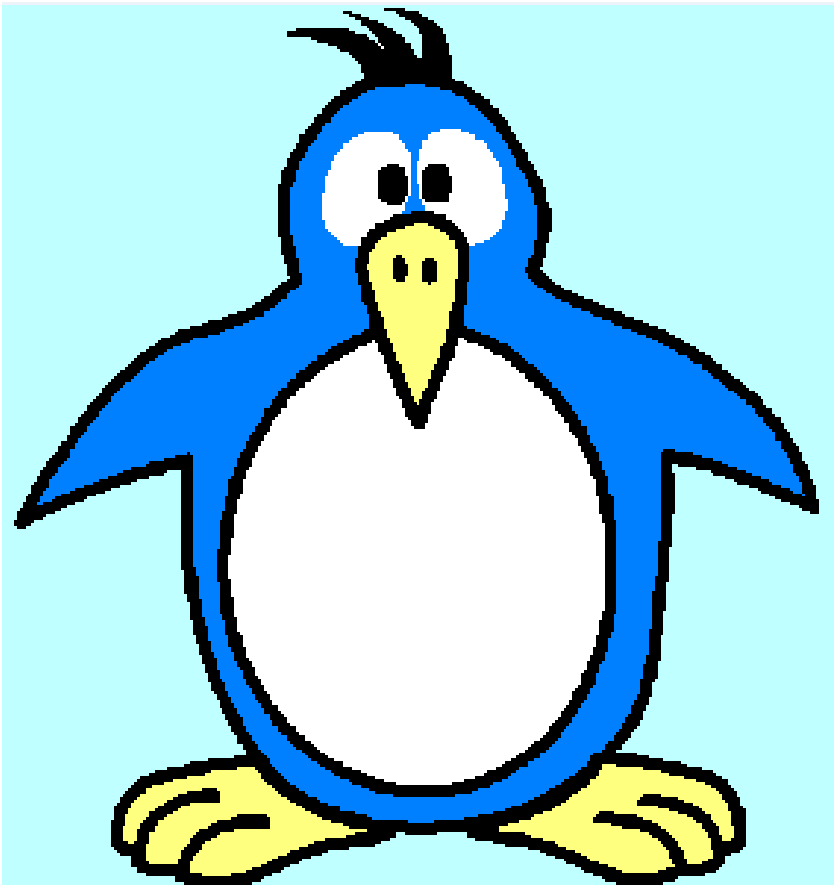

Рисунок 5 — Початкове зображення Bird.bmp

Рисунок 6 — Початкове зображення Boy.bmp

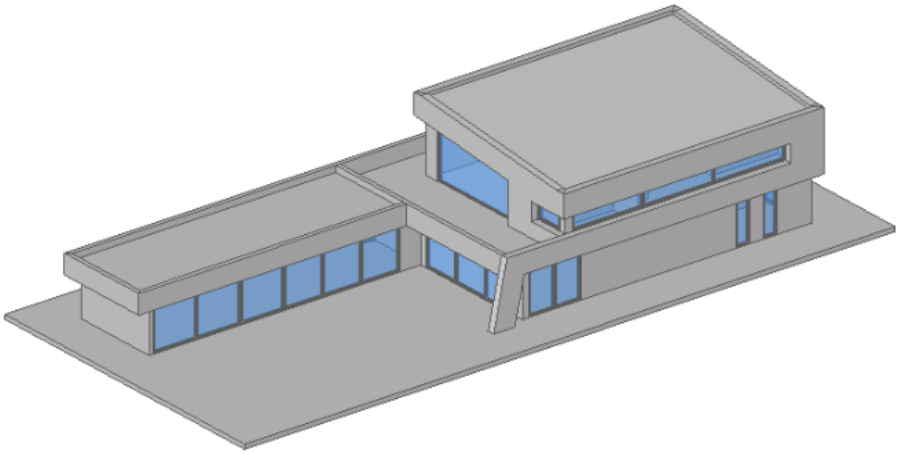

Рисунок 7 — Початкове зображення Pic.bmp

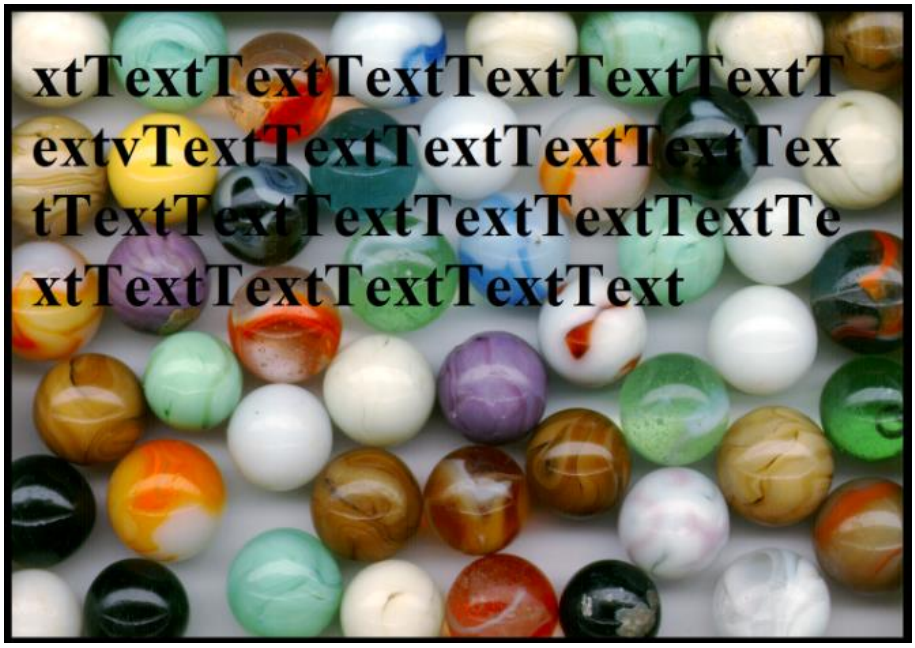


Рисунок 8 — Початкове зображення Tx.bmp

# ДОДАТОК Б. ЛІСТИНГ ПРОГРАМНОГО КОДУ

Для компіляції коду програми необхідно у командному рядку ввести команду «javac Main.java». Створиться файл Main.class. Далі для запуску програми необхідно ввести команду «java Main filename.bmp», де filename.bmp — назва файлу, який конвертується.

**«Main.java»**

```
import javax.imageio.IIOImage;
import javax.imageio.ImageIO;
import javax.imageio.ImageWriteParam;
import javax.imageio.ImageWriter;
import javax.imageio.stream.ImageOutputStream;
import java.awt.image.BufferedImage;
import java.io.File;
import java.util.Iterator;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) throws Exception {
        if (args == null || args.length == 0) {
            System.out.println("Please provide path to source image as first argument!");
            System.exit(1);
        }
        String sourceImagePath = args[0];
        Scanner scanner = new Scanner(System.in);
        String jpegFileName = "./image_from_bmp.jpg";
        String jpegCompressedFileName = "./image_compressed_from_bmp.jpg";
        String gifFileName = "./image_from_bmp.gif";
```

```
            String gifFromJpegFileName = "./image_from_jpeg.gif";
            String gifFromCompressedJpegFileName =
"./image_from_compressed_jpeg.gif";
            {
               System.out.println("Converting source image to JPEG, no
compression.");
               ImageWriter writer = getImageWriter("jpg");
               convertImage(sourceImagePath, writer, jpegFileName, null);
            }
            {
               System.out.println("Converting source image to JPEG with
compression");
               System.out.println("Enter compression quality (0-100): ");
               float quality = scanner.nextInt() / 100f;
               ImageWriter writer = getImageWriter("jpg");
               ImageWriteParam param = writer.getDefaultWriteParam();
               param.setCompressionMode(ImageWriteParam.MODE_EXPLICIT);
               param.setCompressionQuality(quality); // compress to a given quality
               convertImage(sourceImagePath, writer, jpegCompressedFileName,
param);
            }
            {
               System.out.println("Converting source image to GIF");
               ImageWriter writer = getImageWriter("gif");
               convertImage(sourceImagePath, writer, gifFileName, null);
            }
            {
               System.out.println("Converting JPEG image to GIF");
               ImageWriter writer = getImageWriter("gif");
               convertImage(jpegFileName, writer, gifFromJpegFileName, null);
```

```
            }
            {
                System.out.println("Converting compressed JPEG image to GIF");
                ImageWriter writer = getImageWriter("gif");
                convertImage(jpegCompressedFileName, writer,
gifFromCompressedJpegFileName, null);
            }
            System.out.println("Images were converted succesfully.");
        }
        private static void convertImage(String sourceFileName, ImageWriter writer,
String outputFileName, ImageWriteParam param) throws Exception {
            File outputFile = new File(outputFileName);
            ImageOutputStream ios = ImageIO.createImageOutputStream(outputFile);
            writer.setOutput(ios);
            File sourceFile = new File(sourceFileName);
            BufferedImage image = ImageIO.read(sourceFile);
            writer.write(null, new IIOImage(image, null, null), param);
            ios.close();
            writer.dispose();
        }
        private static ImageWriter getImageWriter(String format) {
            // get all image writers for JPG format
            Iterator<ImageWriter> writers =
ImageIO.getImageWritersByFormatName(format);
            if (!writers.hasNext())
                throw new IllegalStateException("No writers found");
            return writers.next();
        }
    }
```